UNIVERZA V LJUBLJANI
FAKULTETA ZA RAČUNALNIŠTVO IN INFORMATIKO

Miha Nedeljko

Programsko določena omrežja v oblaku (SDN)

MAGISTRSKO DELO

ŠTUDIJSKI PROGRAM DRUGE STOPNJE
RAČUNALNIŠTVO IN INFORMATIKA

MENTORICA: doc. dr. Mojca Ciglarič

Ljubljana, 2015

IZJAVA O AVTORSTVU MAGISTRSKEGA DELA

Spodaj podpisani Miha Nedeljko z vpisno številko **63000078**, sem avtor magistrskega dela z naslovom:

Programsko določena omrežja v oblaku.

S svojim podpisom zagotavljam, da:

- sem magistrsko delo izdelal/-a samostojno pod vodstvom mentorice doc. dr. Mojce Ciglarič,
- so elektronska oblika magistrskega dela, naslova (slov., angl.), povzetka (slov., angl.) ter ključne besede (slov., angl.) identični s tiskano obliko magistrskega dela in
- soglašam z javno objavo elektronske oblike magistrskega dela v zbirki »Dela FRI«.

V Ljubljani, dne 26. 6.2015 Podpis avtorja:

Zahvaljujem se mentorici doc. dr. Mojci Ciglarič za strokovno vodenje in usmerjanje ter spodbudo pri izdelavi magistrskega dela. Zahvalil bi se tudi prijateljema dr. Boštjanu Gaberšku in Jerneju Šviglu, ki sta mi s svojim znanjem in strokovnimi nasveti pomagala pri samem študiju.

Iskrena zahvala pa gre moji mami in boljši polovici Andreji, brez katerih ne bi imel toliko motivacije in podpore za dokončanje tega študija.

Kazalo

Kazalo slik

Kazalo tabel

Seznam uporabljenih kratic

kratica	Angleško	Slovensko
IaaS	Infrastructure as a service	Infrastruktura kot storitev v oblaku
PaaS	Platform as a service	Platforma kot storitev v oblaku
SaaS	Software as a Service	Programska oprema kot storitev v oblaku
NaaS	Network as a service	Omrežje kot storitev v oblaku
SDN	Software defined network	Programsko določeno omrežje
OVS	Open virtual switch	Odprto navidezno stikalo
ODL	OpenDaylight controller	Kontrolnik OpenDaylight
OVSDB	Open virtual switch database management protocol	Upravljavski protokol baze odprtega navideznega stikala
CMP	Cloud management platform	Platforma za upravljanje oblaka
OF	Open Flow protocol	Protokol Open Flow
NetConf	Network configuration protocol	Protokol za omrežno nastavljanje
NetFlow	Network Flow protocol	Protokol za zbiranje IP prometa vhodno-izhodnih vmesnikov na stikalu
sFlow	Sampled Flow protocol	Protokol za izvažanje paketov na 2. nivoju OSI omrežnega modela
LLDP	Link Layer Discovery protocol	Protokol za iskanje povezav na fizičnem nivoju
BGP-LS	Border gateway protocol Link-State	Protokol BGP na podlagi stanja povezav
I2RS	Interface to the routing system	Vmesnik do usmerjevalnega sistema
NOS	Network Operating System	Omrežni operacijski sistem
OSI	Open System Interconnection model	Odprti model sistema povezljivosti v računalniškem omrežju
TCP	Transmission Control Protocol	TCP, zanesljiv transportni protokol v omrežjih TCP/IP
MPLS	Multi-Protocol Label Switching	Telekomunikacijski protokol MPLS
MAC	Media access control address	Strojni naslov omrežnega vmesnika, ki ga je dodelil proizvajalec naprave
IP	Internet protocol	IP – omrežni protokol v omrežjih po modelu TCP/IP
IPv4	Internet protocol version 4	Internetni protokol verzije 4
IPv6	Internet protocol version 6	Internetni protokol verzije 6
DNS	Domain name system	Sistem domenskih imen
RPC	Remote procedure call	Protokol za zagon oddaljenih procedur
DHCP	Dynamic Host Configuration Protocol	Omrežni protokol za dinamično nastavitev gostitelja
FQDN	Full qualified domain name	Popolno kvalificirano ime domene
ONF	Open Networking Foundation	Fundacija za odprta omrežja
QoS	Quality of Service	Kakovost storitve
VLAN	Virtual Local Area Network	Navidezno lokalno omrežje
PBB	Provider Backbone Bridge	Most v ponudnikovo hrbtenico. Protokol uporabljajo ponudniki telekomunikacijskih omrežij.

TEP	Tunnel End Point	Zaključek tunela
TTL	Time-to-Live	To je število skokov, ki opredeljuje življenjsko dobo paketa v omrežju.
REST	Representational State Transfer	Spletna arhitektura reprezentacije stanj prenosa spletnih zahtev
JSON	Java Script Object Notation	Notacija objektov JavaScript
SNMP	Simple network management protocol	Preprost protokol za upravljanje omrežij
VRRP	Virtual Routing Redundancy protocol	Protokol za navidezno usmerjanje z redundanco
OSGi	Open Server Gateway Initiative or an open standard organizator	Standard za storitveno platformo, ki zagotavlja celotni in dinamični komponenti model za Javo
YANG	Yet Another Next Generation	Podatkovni model YANG
JVM	Java virtual machine	Okolje, v katerem tečejo javanske aplikacije
SAL	Service abstraction Layer	Raven abstrakcije storitev
AD-SAL	API driven SAL	API usmerjen SAL
MD-SAL	Model-Driven SAL	Modelno usmerjen SAL
STP	Spanning Tree Protocol	Protokol za preprečevanje omrežnih zank
SNBI	Secure Network Bootstrapping Infrastructure	Varnosti protokol SNBI
SDNI	SDN Interface (Cross-Controler Federation)	SDN-vmesnik za namene kontrolnikov v federacijo
FRM	Forwarding router manager	Upravitelj posredujočega usmerjevalnika
AAA	Access, Authorization and Accounting	Dostop, avtorizacija in beleženje porabe
GBP	Group Based Policy	Skupinsko orientirana politika
PCEP	Path Computation Element communication protocol	Komunikacijski protokol za izračun potnih elementov
DDoS	Distirbuted Denial of Service	Porazdeljen napad z zavrnitvijo storitve
SFC	Service Function chaining	Veriženje storitvenih funkcij
PCMM	Packet Cable Multimedia	Protokol za prenos multimedijskih podatkov na kabelskem priključku
DLUX	OpenDaylight User Experience	OpenDaylight uporabniška izkušnja
IT	Information technology	Informacijska tehnologija
KVM	Kernel virtualized machines	KVM hipervizor
Xen	Xen hypervisor	Xen hipervizor
ESXi	VMware hypervisor	VMware hipervizor
vCenter	Virtual center	VMware navidezni centralni strežnik
QEMU	Quick Emulator	Hitri posnemovalnik
API	Application programming interface	Vmesnik za aplikacijsko programiranje
IDS	Intrusion detection system	Sistem za detekcijo napadov
IPS	Intrusion prevention system	Sistem za preprečitev napadov
VPN	Virtual private network	Navidezno zasebno omrežje
LVM	Logical Volume Manager	Tip logične particije na fizičnem disku
SSH	Secure shell	Varnostni dostop do terminala (lupine)
BASH	Bourne again shell	Unix/Linux lupina

AMQP	Advanced massaging queuing protocol	Sporočilni protokol z naprednimi čakalnimi vrstami
UUID	Universally unique identifier	Univerzalni enolični identifikator
VTN	Virtual tenant network	Navidezno omrežje najemnika oblaka
GUI	Graphical user interface	Grafični uporabniški vmesnik
MOM	Message-oriented middleware	Sporočilno orientiran sistem

Povzetek

Danes je na trgu veliko ponudnikov oblačnih storitev, pri katerih lahko najamemo celotna podatkovna središča. Vsi ponudniki oblačnih storitev ponujajo predvsem storitve, ki so vezane na strežniško infrastrukturo. Takim oblačnim storitvam pravimo infrastruktura kot storitev (IaaS). Omogočajo oblikovanje storitev strežniške infrastrukture po meri, ne zagotavljajo pa enakih možnosti pri oblikovanju omrežne infrastrukture. Razlog za to je predvsem v tem, da večina oblačnih storitev še vedno temelji na principu klasične omrežne arhitekture. Tako se poraja potreba po novi arhitekturi, ki bi omogočala prilagajanje omrežja glede na individualne potrebe po oblačnih storitvah. Za zagotavljanje takih storitev moramo razširiti arhitekturo oblaka, tako da ji dodamo arhitekturo programsko določenega omrežja (SDN).

Integracija arhitekture programsko določenega omrežja v obstoječo oblačno arhitekturo je še vedno v fazi razvoja, zato je takšnih ponudnikov storitev malo. Razlog za to so tudi proizvajalci programske in strojne opreme, ki so še vedno nagnjeni h klasičnemu modelu omrežne arhitekture. S tega naslova ponudniki oblačnih storitev uporabniku že v naprej pripravijo določeno topologijo omrežja, ki jo nato uporabnik uporabi pri postavitvi svojih infrastrukturnih storitev, s tem pa ga omejijo v njegovem oblačnem okolju.

V tem magistrskem delu smo predlagali rešitev, ki je pokazala, kako lahko razširimo oblačni arhitekturni model IaaS in mu dodamo arhitekturo programsko določenega omrežja (SDN). Analizirali smo nekaj najbolj znanih oblačnih rešitev in nato predlagali najbolj ustrezno. Enako smo naredili pri kontrolniku programsko določenega omrežja in nato z nekaj primeri uporabe programsko določenega omrežja v praksi pokazali uspešnost delovanja take postavitve.

Za predlagano rešitev smo izbrali oblačno platformo OpenStack in jo razširili s funkcionalnostjo programsko določenega omrežja z integracijo kontrolnika OpenDaylight. Nato smo z aplikacijo na kontrolniku OpenDaylight pokazali njeno uporabo. S tem smo dosegli cilj magistrske naloge: Pokazali smo delovanje rešitve v praksi in tudi njeno uporabnost z vidika varnosti oblačnih storitev. Ugotovili smo, da so nastavitve prikazane rešitve zelo kompleksne, in se tudi zavedamo, da se bo morala tovrstna tehnologija še nekoliko razviti, preden preide v množično uporabo.

Ključne besede: oblačna platforma OpenStack, programsko določena omrežja ali SDN, kontrolnik ODL ali OpenDaylight, odprto stikalo ali OVS.

Abstract

These days there is a large number of cloud service providers on the market where one can rent entire data centers. Cloud providers tend to offer services based on their server infrastructure. This service model is called Infrastructure as a Service. They enable us to design customized server infrastructure services but do not provide for similar possibilities when designing network infrastructure. The reason for this is mostly classical network architecture which is still used in cloud environments. Because of growing demands after flexibility in network area a new approach is needed which suits the cloud services. To achieve this kind of service we need to expand cloud architecture to include the architecture of software defined networks (SDN).

The process of integrating a cloud infrastructure with software defined network is still in development. There are only a few cloud providers that offer these kinds of services. The reason for this are mostly software and hardware providers who still rely on the classical network architecture. Because of this most cloud providers offer predefined network topologies and thus limit customers regarding topology in their cloud infrastructure.

We proposed a solution that has shown how IaaS cloud architecture model can be expanded with the functionality of software defined networks (SDN). We came up with this solution by researching some existing cloud solutions and selecting the most suitable. Same was done for the software defined network solution. We have also prepared some use cases to show how our proposed solution works in practice.

For the proposed solution we used OpenStack cloud platform, which we then extended with the functionality of SDN network by integrating the OpenDaylight controller. We then built a test environment where we demonstrated practical use with the application on the ODL controller. With this we have achieved the goals of master theses and its results in practice. We have also demonstrated its usefulness with regards to cloud security. We concluded that this kind of solutions are very complex and also acknowledge that the technology used for this solution is still in stages of development and needs to evolve more to be used in a production environment.

Keywords: cloud platform OpenStack, Software Defined Network or SDN, OpenDaylight or ODL controller, Open Virtual Switch.

1 Uvod in cilji

Računalništvo v oblaku je za mnoge še vedno neopredeljiv pojem, čeprav se velika večina vsak dan srečuje vsaj z eno storitvijo ali produktom, ki bi ga lahko umestili v to kategorijo. To je bodisi ogled dnevnoinformativne oddaje ali filmov z zamikom bodisi shranjevanje varnostne kopije slik, video vsebin in različnih dokumentov, ki se nahajajo na pametnem telefonu ali osebnem računalniku. In to je le nekaj primerov storitev, ki jih ponuja računalništvo v oblaku. Storitve računalništva v oblaku pogosto uporabljajo tako največje svetovne kooperacije kot tudi manjša podjetja, zaradi česar je to eden od najhitreje rastočih segmentov industrije informacijskih tehnologij (IT).

Računalništvo v oblaku je izraz za opisovanje različnih računalniških konceptov, ki vsebujejo veliko število računalnikov, povezanih prek globalnega omrežja, kot je internet. Pri tem je treba poudariti, da so računalniški koncepti kot medsebojne povezave zelo odvisni od oblačne arhitekture, po kateri so modelirani. Običajno pri računalništvu v oblaku govorimo o treh storitvenih modelih; to so infrastruktura kot storitev (IaaS), platforma kot storitev (PaaS), programska oprema kot storitev (SaaS). Infrastrukturo kot storitev se skoraj vedno ponuja z uporabo tehnologije virtualizacije, kar pomeni, da računalniške vire (infrastrukturo) virtualiziramo, tako da postanejo navidezni. V to kategorijo sodijo strežniki, podatkovni centri, omrežje in podobno. Med vodilnimi ponudniki teh so VMware, Oracle, IBM, Microsoft, Open Stack, Eucalyptus, Citrix Cloud in Amazon EC2. Platforma kot storitev sodi v okolje za razvoj aplikacij ali platform. Na tem nivoju se uporabnikom ni treba ukvarjati z računalniško infrastrukturo. Vodilni ponudniki tovrstnih storitev so Google AppEngline, Microsoft Azure, Amazon AWS in nekateri drugi. V programsko opremo kot storitev sodijo funkcionalnosti poslovnih aplikacij ali programska oprema. Je najvišji nivo oblačne arhitekture. Na tem nivoju je uporabnikom zagotovljena celotna infrastruktura skupaj s programsko opremo in nastavitvami za njeno delovanje. Med ponudniki na tem področju so danes najbolj razširjeni Google App, Microsoft 365, SalesForce CRM in nekateri drugi.

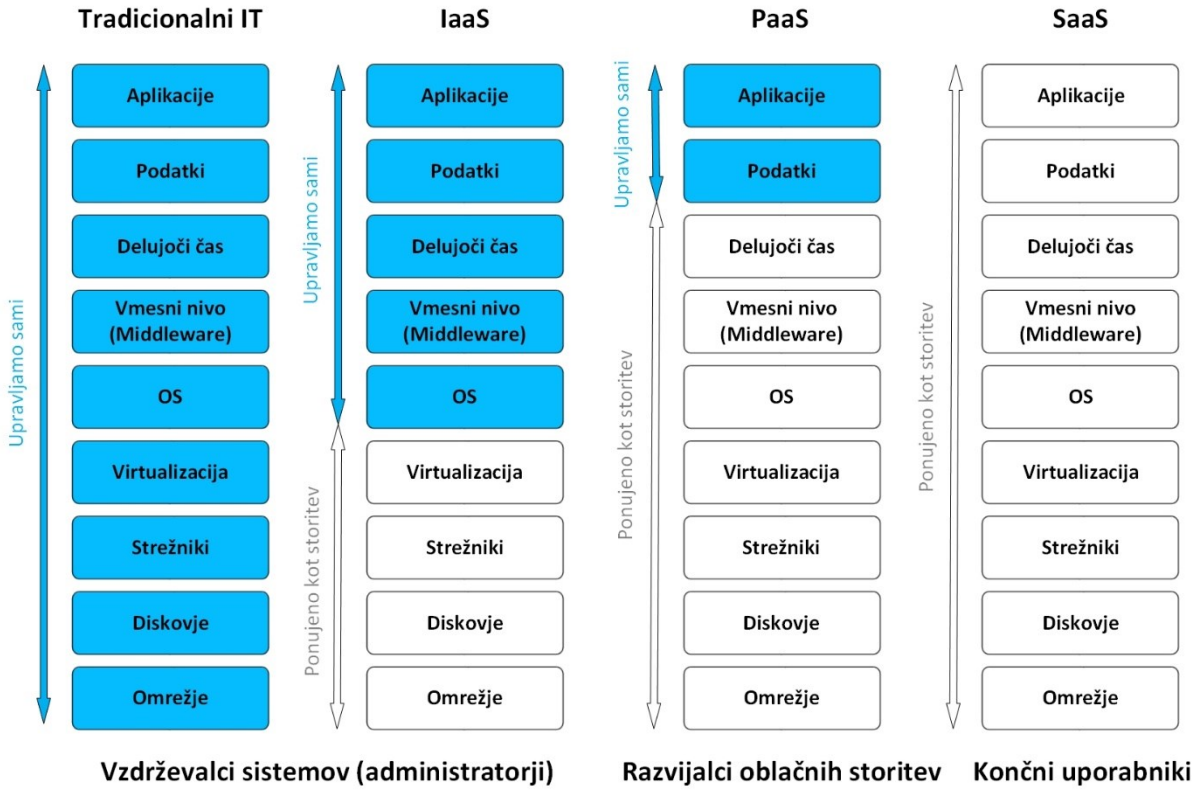
Cilj magistrskega dela je definirati oblačno arhitekturo, ki nudi oblačne storitve na nivoju IaaS in bo vključevala integracijo programsko določenega omrežja (SDN), ne bomo pa se ukvarjali z zgornjimi plastmi oblačne arhitekture na nivoju PaaS in SaaS. Nato bomo arhitekturo oblaka dopolnili tako, da bo uporabniku omogočala prilagajanje omrežja glede na potrebe aplikacije. To bomo dosegli tako, da bomo razširili obstoječo oblačno arhitekturo infrastrukturnih storitev z vključitvijo izbranega SDN-kontrolnika. Doseganje ciljev bomo prikazali na praktičnem primeru.

2 Računalništvo v oblaku

Računalništvo v oblaku omogoča vseprisoten in prikladen dostop do omrežja na zahtevo različnim deljenim skupinam nastavljivih računalniških virov (npr. omrežij, strežnikov, hrambe podatkov, aplikacij in storitev), ki jih lahko hitro oskrbujemo in izdamo z minimalnim naporom upravljanja ali interakcijo ponudnika storitev. Računalništvo v oblaku teče na zanesljivih strežnikih ali različnih računalniških centrih, ki se lahko nahajajo povsod po svetu in so uporabnikom oblaka vedno dostopni na daljavo. Oblačna arhitektura je razdeljena na tri dele, ki skupaj tvorijo oblačno piramido: spodnji nivo predstavlja infrastrukturo kot storitev v oblaku (IaaS), srednji nivo predstavlja platformo kot storitev v oblaku (PaaS), zgornji nivo pa predstavlja programsko opremo kot storitev v oblaku (SaaS) [40].

Računalništvo v oblaku spreminja način poslovanja in razmišljanja organizacij v gospodarstvu. Organizacija IDC napoveduje izjemno rast globalnih ponudnikov oblačnih storitev. Posledično naj bi se krog oblačnih ponudnikov storitev IaaS na globalni ravni v prihodnje vse bolj ožil. A računalništvo v oblaku po drugi strani omogoča hitro trženje računalniških storitev zaradi svoje rastoče poslovne agilnosti. Gospodarske organizacije lahko zato z računalništvom v oblaku enostavno eksperimentirajo in zelo hitro razvijajo inovacije ter tako prilagajajo nove rešitve brez velikih začetnih finančnih investicij.

Oblak tako postaja dobro vezivo med različnimi sistemi in različnim obnašanjem. Računalništvo v oblaku spreminja način, kako uporabniki uporabljajo tehnologijo, in tako kreira nove oblike oblačnih potrošnikov (potrošniki na različnih nivojih arhitekture oblaka, kot so IaaS, PaaS, SaaS). S tem pa gospodarske organizacije pridobijo več inovativnosti pri ponudbi storitev v svojih poslovnih modelih. Tako je v zadnjih letih zraslo število zasebnih in javnih oblakov, saj z njihovo pomočjo zmanjšujemo stroške in hitreje pripravimo aplikacije za končno uporabo. Organizacije tudi ne potrebujejo več velikega števila oseb, ki vzdržujejo podatkovna središča, temveč se lahko osredotočijo na razvoj samih aplikacij.



Slika 2.1: Odgovornost računalničarjev glede na posamezne oblačne arhitekture

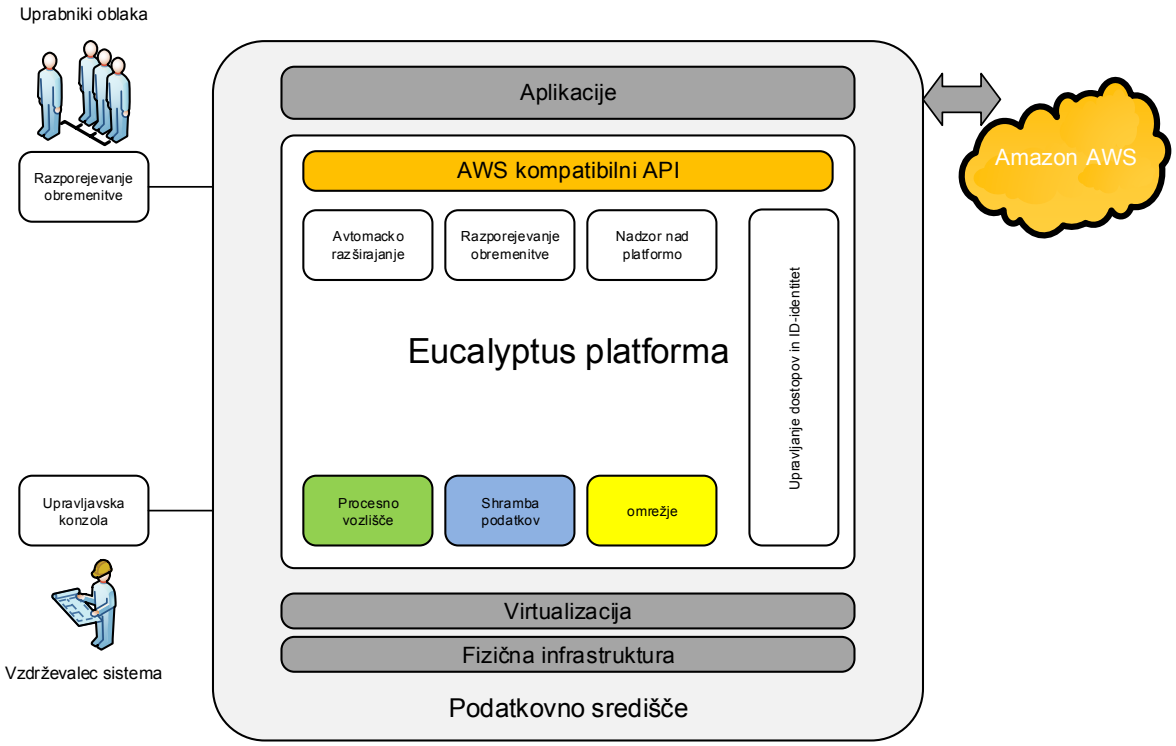
Na sliki 2.1 lahko opazimo, kako se spreminja razmejitev odgovornosti posameznega IT-osebja, ko se vzpenjamo po oblačnih arhitekturah. Z modro barvo je predstavljena odgovornost IT-osebja. Z vidika zasebnih in hibridnih oblačnih platform je zanimiva predvsem oblačna arhitektura IaaS, PaaS in SaaS, ki pridejo v poštev pri javnih ponudnikih oblačnih storitev. Pomembno je tudi, v kolikšnem obsegu bo uporabnik želel uporabljati storitev oblačne platforme ponudnikov, kar pa se dogovori na ravni storitve (SLA).

V magistrskem delu smo se osredotočili pretežno na arhitekturni model IaaS in temu primerno odprtokodno oblačno platformo. Pri izbiri oblačne platforme se je pojavilo kar nekaj dilem, saj je na trgu veliko rešitev oblačnih platform, v magistrskem delu pa smo izbrali štiri rešitve, ki so zanimive in so najbolj prisotne v gospodarstvu ter odprtokodni skupnosti. To so Eucalyptus, OpenNebula, CloudStack in OpenStack.

2.1 Oblačna platforma Eucalyptus

Oblačna platforma Eucalyptus je odprtokodna oblačna platforma za grajenje zasebnih in hibridnih oblakov. Hibridni oblak omogoča najemniku storitev, da uporablja računalniške vire v obeh; v zasebnem in v javnem oblaku. Trenutni lastnik projekta je podjetje HP in ga je integriralo v svoje oblačne storitve (HP Helium Cloud). Osredotoča se na upravljanje oblačnega arhitekturnega modela IaaS. Zelo je podoben najbolj znanemu ponudniku oblačnih storitev Amazon AWS in se zato dobro povezuje z njihovimi oblačnimi storitvami. Obstajata dve različici: odprtokodna in komercialna. Odprtokodna različica je omejena s podporo upravljanja infrastrukture za virtualizacijo (podpira odprtokodne različice

hipervizorja KVM in Xen). Nekaj več podpore ima komercialna različica, ki podpira VMware ESXi hipervizor. Hipervizor je virtualizacijsko jedro, ki obvladuje in zagotavlja delovanje navideznih računalnikov v navideznem okolju.



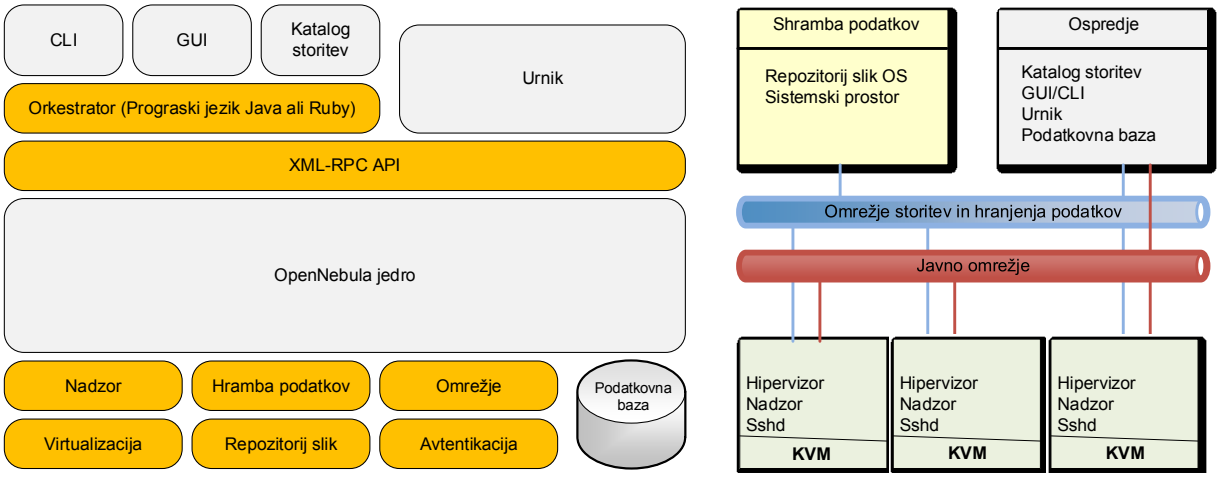
Slika 2.2: Pregled nad oblačno arhitekturo Eucalyptus

Slika 2.2 prikazuje oblačno arhitekturo Eucalyptus. Platforma zagotavlja upravljanje navideznih računalnikov na platformi Eucalyptus in instanc Amazon (navideznih strežnikov, ki se izvajajo v javnem oblaku). Uporabnikom oblaka omogoča selitev navideznih strežnikov iz privatnega v javni oblak, s tem pa zagotavlja prilagodljivost privatnega oblaka (hibridnost). Platformo Eucalyptus sestavlja šest glavnih komponent. Upravljalna komponenta platforme je oblačni kontrolnik, ki služi za upravljanje, optimizacijo in rezervacijo zahtev, ki prihajajo na vmesnik API preko upravljavske konzole, in tako upravlja spodnje procesorske, hranilne in omrežne komponente. Omogoča tudi kompatibilni vmesnik EC2, ki zagotavlja povezljivost privatnega oblaka z javnim oblakom Amazon, in hranjenje podatkov kot storitev, ki delujejo po principu spletne arhitekture REST. Gručni kontrolnik platforme skrbi za visoko razpoložljivost oblačnih virov. Za usklajevanje in blokovni zapis na oblačni platformi skrbi diskovni kontrolnik, ki komunicira z oblačnim in gručnim kontrolnikom. Vozliščni kontrolnik skrbi za nadzor in upravljanje navideznih strežnikov ter celotnega omrežja. V komercialni verziji platforme Eucalyptus imamo še dodatno komponento VMware broker, ki skrbi za integracijo z navidezno infrastrukturo VMware [1,2,3,7].

Predvsem pri tej oblačni platformi pogrešamo odprtost API-vmesnikov, ki so pri drugih oblačnih platformah bolj popolni. Tako smo za razširitev zasebnega oblaka omejeni na izbiro javnih oblakov. V primeru te oblačne platforme je razširitev mogoča samo na javni oblak Amazon.

2.2 Oblačna platforma OpenNebula

Oblačna platforma OpenNebula je odprtokodna platforma, ki je usmerjena v oblačni arhitekturni model IaaS. Tako kot platforma Eucalyptus obstaja v odprtokodni in komercialni različici. Platforma podpira več različnih hipervizorjev, kot so na primer KVM, Xen, VMware ESXi. Zna se integrirati z različnimi oblačnimi ponudniki, kot so Microsoft Azure, Amazon EC2 in IBM SoftLayer. S tem nudi dobro razširitev privatnega oblaka v javni oblak. Oblačna platforma je definirana s tremi glavnimi komponentami; to so shramba podatkov, omrežje in hipervizorji. Slika 2.3. prikazuje arhitekturo oblačne platforme OpenNebula. Vse storitve platforme se izvajajo na gostitelju, ki teče v ospredju in nato komunicira preko spletne (omrežne) storitve z vsemi hipervizorji. Tako oblačna platforma kreira, nadzira in upravlja z navideznimi računalniki, ki tečejo na hipervizorjih.



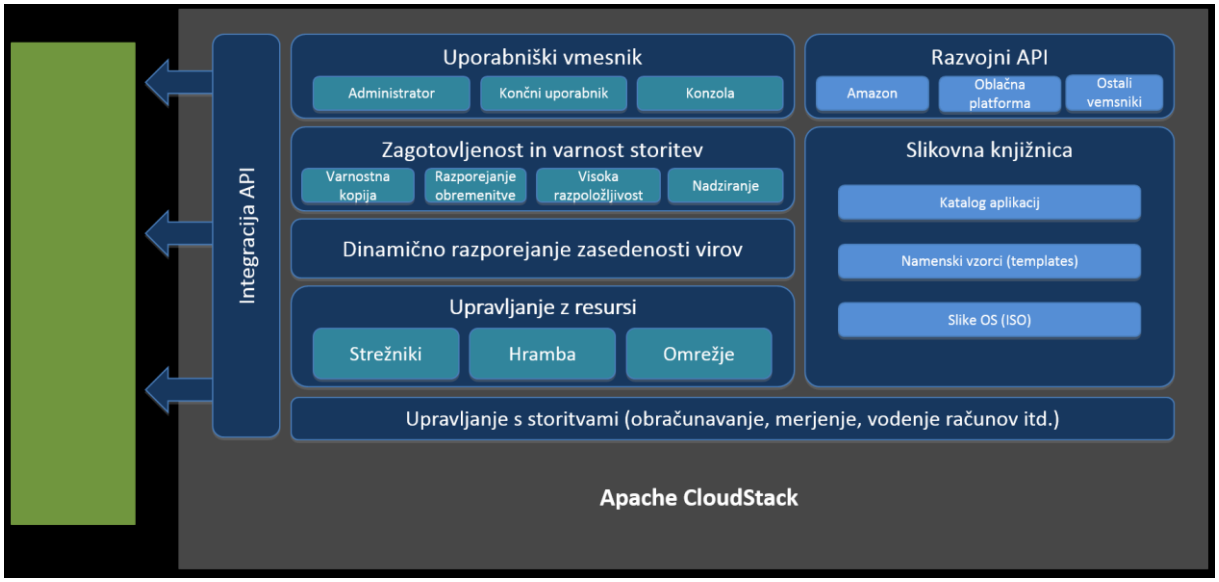
Slika 2.3: Pregled arhitekture oblačne platforme OpenNebula

Komponenta za shranjevanje podatkov preko omrežja za shrambo podatkov zagotavlja diskovni prostor in deponijo slik navideznih strežnikov, ki tečejo na hipervizorjih. Navidezni strežniki sodijo bodisi v zasebno bodisi v javno omrežje in se povezujejo z navideznimi omrežji VLAN (logična skupina omrežnih vozlišč, ki predstavljajo navidezno omrežje LAN). Posamezno omrežje VLAN lahko pripada enemu ali več uporabnikom kot tudi eni ali več skupinam uporabnikov. Navidezni strežniki komunicirajo preko izoliranega privatnega omrežja v privatni postavitvi oblaka in preko javnega omrežja, ko želimo postaviti hibridni oblak. Za visoko razpoložljivost oblačne platforme poskrbi gručni sistem, ki zagotavlja, da ne pride do izpada oblačnih storitev. Kot pri oblačni platformi Eucalyptus so gručni sistemi del komercialne verzije oblačne platforme. OpenNebula zagotavlja uporabnikom elastično platformo za hitro dostavo oblačnih storitev [2,4,7].

Zaradi enostavne in hitre uporabe pa je platforma omejena na nivoju prilagodljivosti razširitve oblaka na druge navidezne ali oblačne platforme (oblačna platforma ne sodeluje dobro z drugimi oblačnimi platformami).

2.3 Oblačna platforma CloudStack

CloudStack je odprtokodna oblačna platforma, ki jo je razvilo podjetje Cloud.com pod licenco GNU (licenca, ki omogoča prosto distribuiranje in predelovanje vsebine). Pred časom ga je prevzelo podjetje Citrix in ga podarilo fundaciji Apache Software (ASF). Kot že opisani oblačni platformi se tudi CloudStack osredotoča na arhitekturni model oblaka IaaS. Podpira veliko različnih hipervizorjev, kot so na primer VMware ESXi, Xen, OracleVM, KVM in Microsoft HyperV.



Slika 2.4: Arhitektura oblačne platforme CloudStack

Uporabniki lahko upravljajo z oblačno infrastrukturo preko spletnega vmesnika ali orodne vrstice v konzoli. Na voljo pa imajo vmesnike RESTful API, ki so dobro dokumentirani. Z vmesniki API se lahko povežemo v javni oblak ponudnika Amazon in tako tvorimo hibridni oblak. Platforma s tem podpira razširitveno arhitekturo. Omogoča visoko razpoložljivost oblačnih virov s pomočjo razporejanja obremenitev po različnih vozliščih v oblaku. Omogoča tudi sposobnost kreiranja navideznih omrežij in podvojevane (razmnoževanje – replication) podatkov centralne podatkovne baze.

Za razliko od prej opisanih oblačnih platform lahko pri tej uporabljamo vse funkcionalnosti v obeh različicah oblačne platforme (odprtokodni in komercialni) [3,5,8].

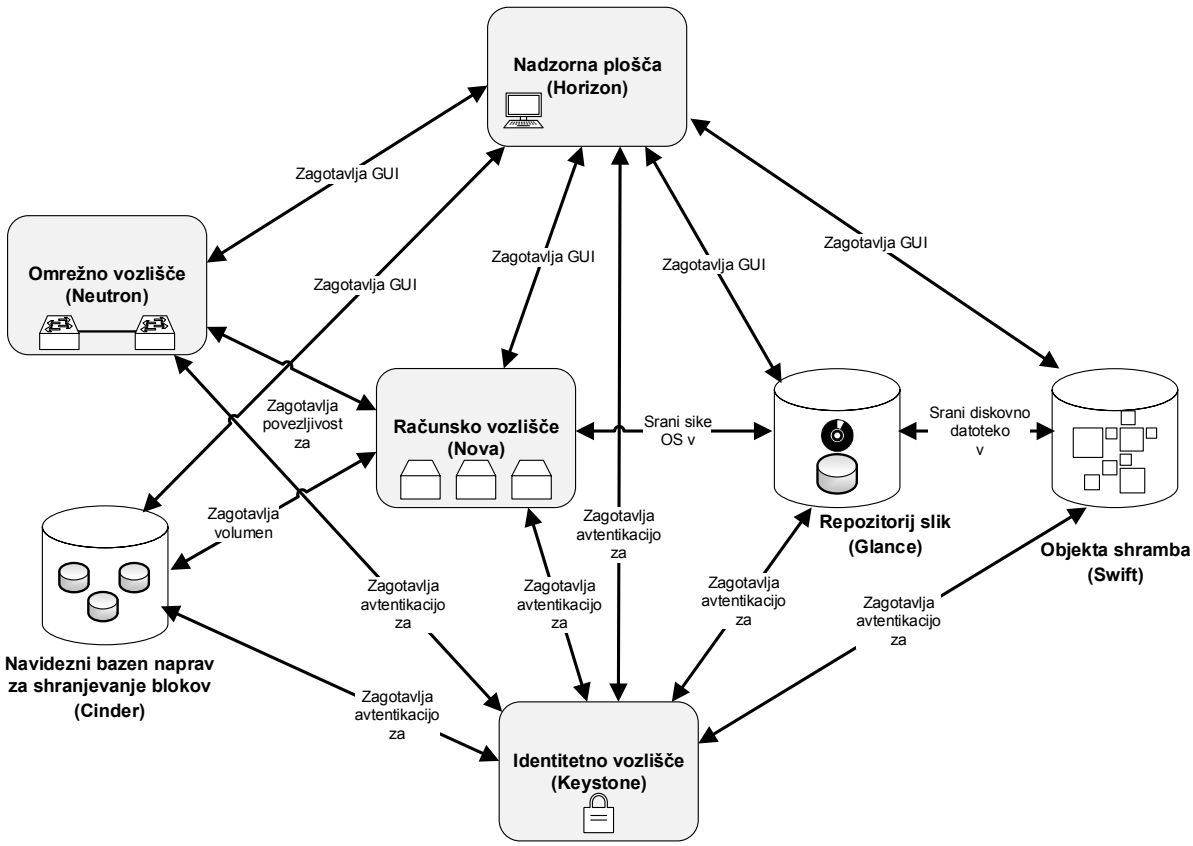
2.4 Oblačna platforma OpenStack

OpenStack je trenutno vodilna odprtokodna oblačna platforma, ki ponuja zelo dobro razširitev privatnih in javnih oblačnih storitev po oblačnem arhitekturnem modelu IaaS. Ima veliko podporo v odprtokodni skupnosti in je močno prisotna v gospodarskih (RedHat, IBM, Intel, Rackspace itd.) kot tudi v raziskovalnih vodah (CERN, NASA, MIT, CESNET, IMOS itd.). Omenjene organizacije uporabljajo

OpenStack kot glavno oblačno platformo, na kateri temelji večina njihovih storitev, ter so tako vezane in motivirane k nadaljnjemu razvijanju te oblačne platforme. Zaradi tega lahko predvidevamo, da bo oblačna platforma v gospodarstvu v bodočnosti še močno prisotna.

Oblačna platforma OpenStack omogoča uporabnikom kreiranje navideznih strežnikov in drugih instanc (navideznih elementov omrežja), ki opravljajo različne naloge upravljanja v oblačnem okolju med samim delovanjem. Tako zagotavlja hitro in enostavno horizontalno širjenje oblačnih virov. Horizontalno širjenje oblačnih virov pomeni, da dodajamo nove navidezne strežnike, ki imajo identične nastavitve kot že obstoječi virtualni strežniki v oblaku. Razporejanje obremenitve oblačnih storitev se tako porazdeli na nove in že obstoječe skupine virov. Takemu širjenju pravimo tudi širjenje navzven (scaling out). Zagotavljanje take infrastrukture pomeni, da uporabniki platforme OpenStack zelo enostavno dodajajo nove instance, na katerih poganjajo svoje storitve, ki lahko služijo kot oblačne komponente. Širjenje oblačnih virov poteka po potrebi dinamično z večjim številom istih instanc, po katerih se razporedijo uporabniške storitve med samim delovanjem [6].

Za primer lahko vzamemo mobilno aplikacijo, ki komunicira z oddaljenim strežnikom. Med povečevanjem števila uporabnikov se po potrebi kreirajo nove instance strežniške aplikacije, ki komunicirajo med seboj in tako razbremenijo druge obstoječe instance.



Slika 2.5: Arhitektura oblačne platforme OpenStack

Na sliki 2.5 so prikazane osnovne komponente oblačne platforme OpenStack in njihov medsebojne relacije. Platformo sestavlja sedem glavnih komponent, ki opravljajo posamezne oblačne funkcionalnosti. Na sliki 2.5 je vidna tudi relacija med njimi. Komponente, ki sestavljajo oblačno platformo, so sledeče [6,7]:

a) Keystone je komponenta, ki služi kot OpenStack upravnik identitete. Služi kot skupni avtentikacijski in avtorizacijski sistem za celotno oblačno platformo. Uporablja se za avtentikacijo in avtorizacijo klicev do vmesnika API, ko komunicira z ostalimi OpenStack servisi. Podpira avtentikacijo, ki temelji na žetonih (tokens), in avtorizacijo na nivoju servisnih uporabniških profilov [6].

b) Nova je komponenta, ki upravlja procesiranje virtualnih računalnikov. Zagotavlja kreiranje, upravljanje, nadziranje in avtomatizacijo instanc (navideznih strežnikov) gručnih virov oblaka. Podpira širok spekter virtualnih tehnologij, kot so različni hipervizorji QEMU, KVM, Xen, Citrix XenServer, VMware ESXi in vCenter, HyperV in LXC. Vsebuje tudi vmesnik API za integracijo z javnim oblakom Amazon EC2 in S3 [6]. Novo bomo v nadaljevanju poimenovali kot računsko vozlišče.

c) Neutron je komponenta, ki zagotavlja upravljanje omrežij na oblačni platformi. Neutron skrbi, da omrežje ne postane omejitev ali ozko grlo na omrežnem nivoju z zagotavljanjem storitev, povezanih s podrobnimi nastavitvami omrežja. Uporablja se za kreiranje lastnih uporabniških omrežij, kontrolo prometa in povezovanje omrežnih naprav v različna omrežja. Neutron zagotavlja omrežne modele za različne uporabnike in skupine. Standardni model vsebuje navadno ali VLAN-omrežje. Omogoča upravljanje statičnega (statični IP-naslov) ali dinamičnega (IP-naslov se dodeli preko protokola DHCP) naslovnega prostora, pa tudi plavajoče (floating) IP-naslove, ki omogočajo dinamično preusmeritev omrežnega prometa do kateregakoli omrežnega vira v IT-infrastrukturi. Plavajoči (floating) IP-naslov je naslov, ki je dodeljen napravi v oblaku, tako da je lahko dostopna navzven v javnem omrežju (omrežju LAN ali WAN). Ta lastnost je priročna, kadar opravljamo vzdrževalna dela ali v primeru izpadov določenih segmentov na oblaku (preusmeritev med vzdrževanjem ali izpadom omrežja). Poleg vsega naštetega Neutron vsebuje dodaten nabor omrežnih storitev, ki nam zagotavljajo boljše razporejanje (load balancing), varnejše (požarne pregrade, VPN) in bolj nadzorovano (IDS in IPS: sistema za zaznavanje in preprečevanje vdorov v omrežje) omrežje v oblaku [6]. Neutron bomo v nadaljevanju poimenovali omrežno vozlišče.

d) Glance je komponenta, ki zagotavlja storitev skladiščenja slik navideznih strežnikov. Glance omogoča odkrivanje, registracijo in vzpostavitev slik navideznih strežnikov. Vmesnik RESTful API, ki ga vsebuje Glance, nam omogoča poizvedovanje po metapodatkih shranjenih slik, ki so v inventarju skladišča. Slike, ki so dostopne z Glance, se lahko nahajajo na različnih lokacijah – od datotečnih sistemov do objektnih shramb, kot jo zagotavlja komponenta Swift na platformi OpenStack [6].

e) Cinder je komponenta, ki zagotavlja navidezni bazen (pool) naprav za shranjevanje blokov podatkov. Zagotavlja tudi vmesnik RESTful API, preko katerega lahko uporabniki rezervirajo prostor v oblaku, ne da bi se pri tem zavedali, kje se dejansko nahajajo podatki. Uporablja blokovno strukturo LVM, s katero se prezentira kot diskovni prostor. Uporabniki v oblaku ga uporabljajo s pomočjo komponente Nova. LVM je sistem, ki zagotavlja logične particije neodvisno od spodnje postavitve fizičnih diskov. S tem dobimo abstraktni nivo hrambe podatkov, v katerem so navidezne particije, ki jih lahko zelo enostavno krčimo in širimo [6].

f) Objektna shramba (Swift) je komponenta, ki zagotavlja porazdeljen sistem hranjenja podatkov in nudi možnost razširitve na veliko število strežnikov v oblaku. Za shranjevanje s pomočjo

komponente Swift so najbolj primerni nestrukturirani podatki, kot so arhivski, spletni, mobilni in vsi drugi, ki nimajo omejene rasti. Podjetje Rackspace ga uporablja kot glavni sistem za hranjenje podatkov svojih storitev. Swift vsebuje vmesnik RESTful API, preko katerega lahko delamo poizvedbe o hranjenih podatkih [6].

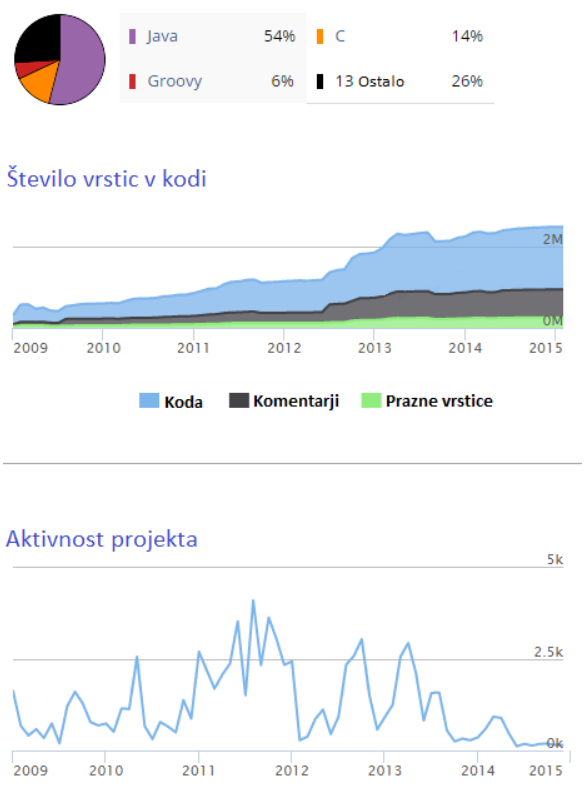
g) Horizon je komponenta, ki zagotavlja nadzorno ploščo in služi kot spletni uporabniški vmesnik za upravljanje večine naštetih komponent oblačne platforme OpenStack.

Na opisane komponente se bomo v naprej sklicevali kot na vozlišča oblačne platforme OpenStack.

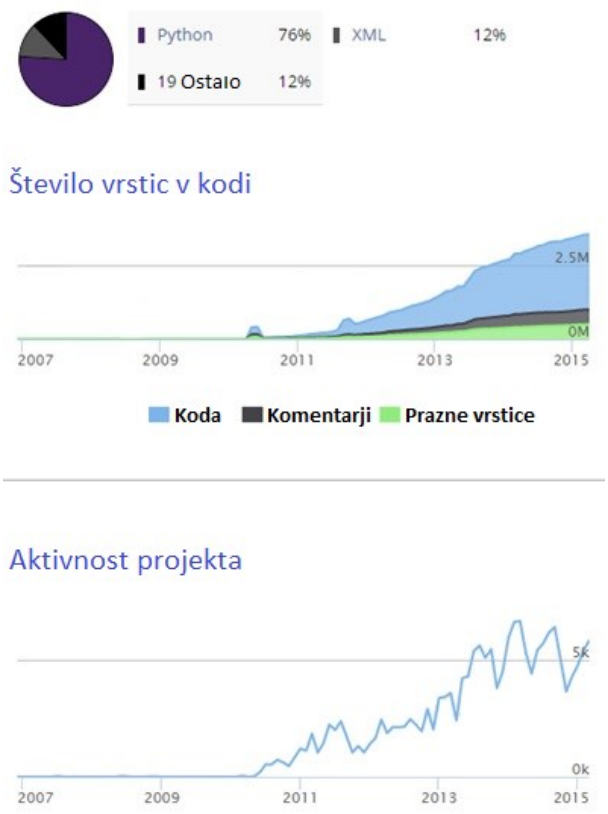
2.5 Izbira oblačne platforme

V tem poglavju smo na kratko predstavili delovanje izbranih odprtokodnih oblačnih platform. Za nadaljnjo implementacijo programsko določenih storitev v oblaku pa smo se odločili za uporabo odprtokodne oblačne platforme OpenStack. Razlog za to je predvsem njena fleksibilnost, prilagodljivost in odprtost, ki je pri oblačnih platformah CloudStack, OpenNebula in Eucalyptus slabša. Najbolj uporabna lastnost platforme OpenStack je njena podrobna prilagodljivost. Prilagodljivost hkrati pomeni večjo kompleksnost in zahtevnejšo postavitev, najzahtevnejšo od vseh opisanih. Dodaten razlog za njeno izbiro je priljubljenost in razširjenost oblačne platforme OpenStack v gospodarstvu in odprtokodni skupnosti. Odprtokodna skupnost je zato živa in zelo aktivna, kot lahko razberemo s statističnih podatkov na slikah 2.6, 2.7, 2.8 in 2.9.

Slika 2.6: Prikaz aktivnosti projekta OpenNebula



Slika 2.7: Prikaz aktivnosti projekta Eucalyptus



Slika 2.8: Prikaz aktivnosti projekta OpenStack

Slika 2.9: Prikaz aktivnosti projekta CloudStack

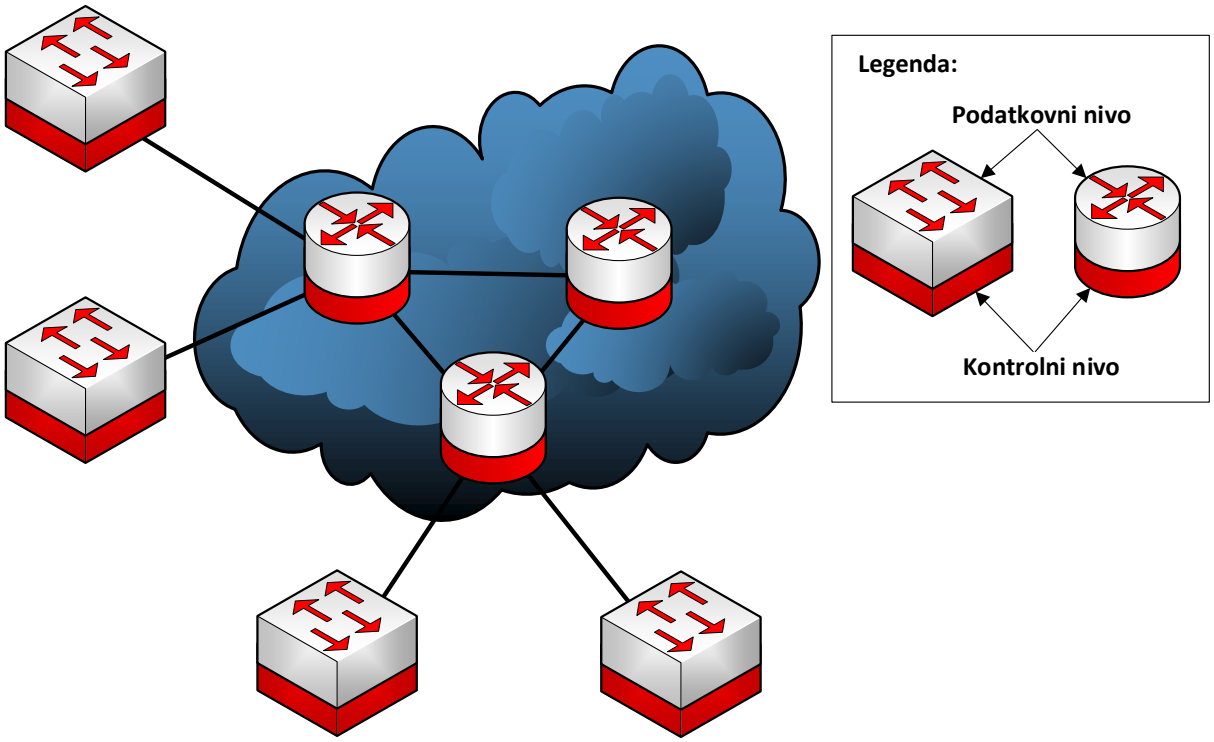
S slik 2.6, 2.7, 2.8 in 2.9 so razvidni statistični podatki prispevane količine vrstic napisane kode razvijalcev oblačnih platform. Zgornji del slik (vertikala, ki predstavlja število vrstic kode) prikazuje rast velikosti kode na projektu, spodnji del slike (vertikala, ki predstavlja aktivnost razvijalcev na projektu) pa interval časovne aktivnosti razvijalcev programske opreme. V obeh delih slike (spodnjem in zgornjem) horizontala predstavlja življenjsko dobo projekta.

Kot lahko vidimo, se v zadnjih treh letih najbolj razvijata oblačni platformi OpenStack in CloudStack, kjer se število aktivnosti razvijalcev programske opreme na projektu povečuje (slika 2.8 in 2.9) in je v nasprotju s številom aktivnosti na projektih OpenNebula in Eucalyptus (slika 2.6 in 2.7). Podatke smo pridobili na spletnem portalu Open Hub (www.openhub.net), ki je v lasti podjetja Black Duck Software in se že vrsto let ukvarja s statistično analizo projektov, ki jih producirajo odprtokodne skupnosti.

3 Programsko določena omrežja (SDN)

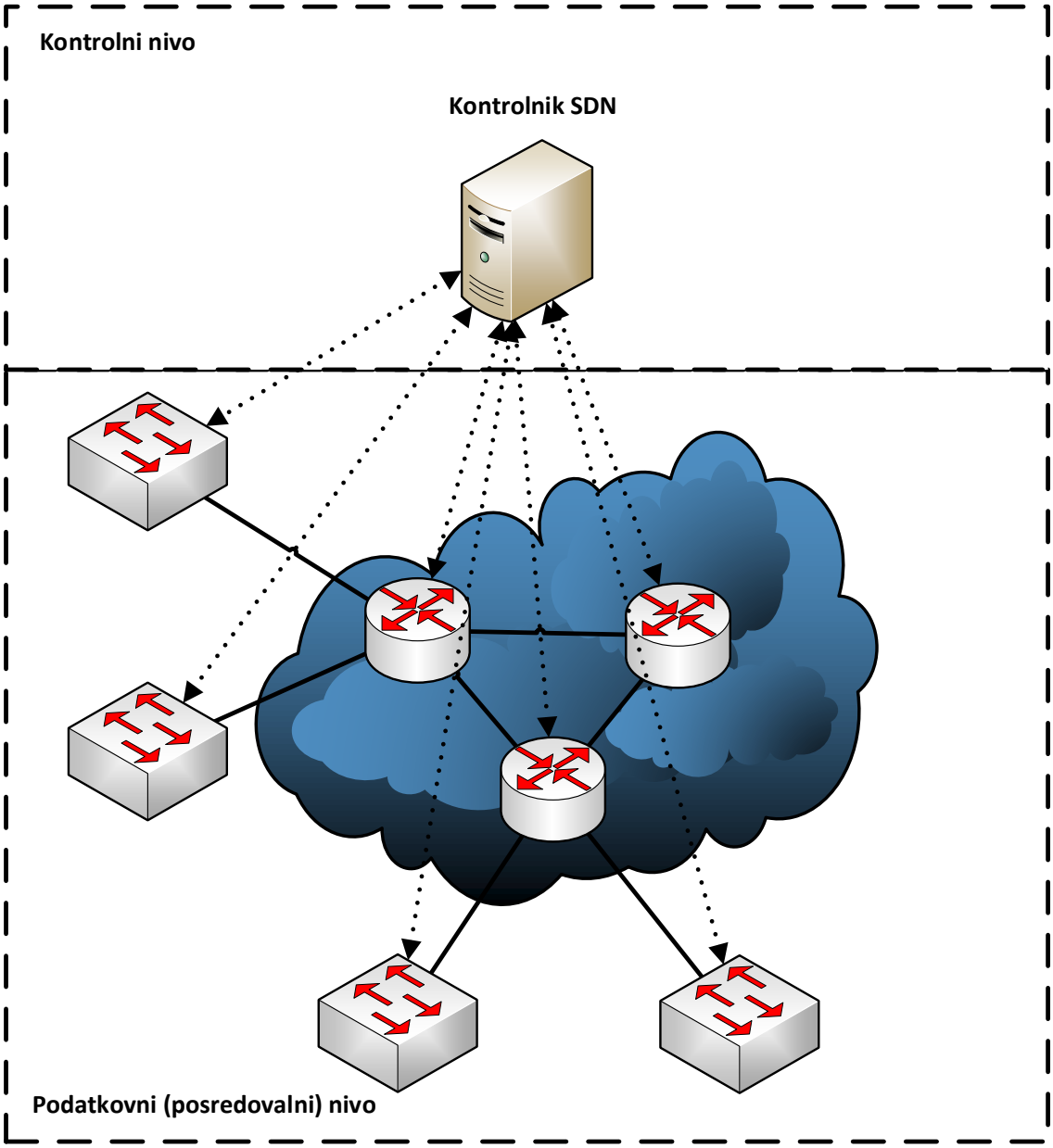
3.1 Kaj je SDN

Pojem programsko določeno omrežje se je prvič pojavil v raziskavah, narejenih na Univerzah Berkeley in Stanford. Od klasičnega računalniškega omrežja, ki ga poznamo, se programsko določeno omrežje razlikuje po tem, da med seboj ločuje kontrolni in podatkovni nivo omrežja. Kontrolni nivo predstavlja logiko, ki je odgovorna za sprejemanje odločitev o tem, kako bo potekal promet celotnega omrežja. Podatkovni nivo pa predstavljajo nizkonivojski omrežni elementi, ki imajo nalogo posredovanja omrežnih paketov od izvora (začetka) do ponora (konca) po navodilih kontrolnega nivoja. V klasičnem računalniškem omrežju sta kontrolni in podatkovni nivo zaklenjena na fizičnem omrežnem stikalu ali usmerjevalniku, številni omrežni protokoli so tako implementirani na kontrolnem nivoju, ki je vgrajen v namensko strojno omrežno opremo. Tako smo omejeni na strojno opremo (omrežne naprave), preko katerih tečejo naši podatki, in moramo za vsako novo omrežno funkcionalnost menjati omrežne naprave.



Slika 3.1: Klasično računalniško omrežje

V programsko določenih omrežjih se kontrolni nivo prestavi v programsko centralizirano kontrolno enoto, ki ga imenujemo kontrolnik SDN (slika 3.1). Ker je kontrolnik SDN narejen kot programska oprema, se lahko izvaja na fizičnem oz. navadnem strežniku, ki zagotavlja več računalniških virov, kot jih zagotavlja namenska strojna oprema omrežnih stikal in usmerjevalnikov. Razširjanje kontrolne logike kontrolnika SDN in s tem celotnega SDN-omrežja je tako enostavno, saj se nadgradnja naredi samo na aplikativnem nivoju brez potrebe menjave strojne omrežne opreme [9].



Slika 3.2: Programsko določeno omrežje

Kontrolo nad tokom podatkov (omrežnih paketov) v programsko določenem omrežju vrši SDN-kontrolnik tako, da elementom v omrežju (stikala, usmerjevalniki itd.) spreminja (posodablja) posredovalne ali tokovne tabele (forwarding Flow Tables). Postopek spreminjanja tokovnih tabel poteka preko odprtega vmesnika, ki je določen s protokolom (npr. OF, NetConf ali OVSDB) [14,15,17,18]. Tudi sporočila, povezana s spremembo topologije omrežja, se pošiljajo preko istega vmesnika. Programsko določena omrežja se poslužujejo protokolov, kot so LLDP, BGP-LS in I2RS [9,10], ki so implementirani v SDN-kontrolniku kot rešitev za iskanje topologije omrežja. Pravila, ki narekujejo posodabljanje tokovnih tabel, se lahko vršijo na proaktivni ali reakcijski način. Proaktivni način posodablja tokovnih tabele s pravili pred prispetjem novega omrežnega paketa, reaktivni način pa primerja novo prispele tokove s tokovnimi pravili v tabeli, in če ne ustrezajo nobenem izmed tokovnih pravil v tabeli, jih pošlje SDN-kontrolniku v procesiranje. SDN-kontrolnik procesira neustrezni tok in posodobi pravila v tokovnih tabelah na omrežnem stikalu ali usmerjevalniku. Vsako omrežno napravo

v programsko določenih omrežjih lahko tako imamo za neumno napravo, ki ima enostavno nalogo proženja akcije nad omrežnim paketom, ki jo narekuje njegova tokovna tabele (obstajajo tudi hibridna stikala in usmerjevalniki, ki lahko v primeru izpada povezave do kontrolnika SDN preko lastne integrirane logike nadaljujejo procesiranje omrežnih paketov). Hitrost pošiljanja med omrežnimi napravami je odvisna od hitrosti pregledovanja tokovnih tabel, kar pa pogojujejo pravila OF, nameščena na omrežni napravi [9].

3.2 Motivacija uporabe SDN

Programsko določena omrežja fundamentalno spreminjajo načrtovanje, način izgradnje in operativnost klasičnih računalniških omrežij. Dandanes podatkovna središča postajajo vse bolj dinamična in se prilagajajo potrebam po računalniških virih. SDN-omrežja poskušajo enak koncept zagotoviti tudi na omrežni infrastrukturi in dinamično zagotavljajo spremembe glede na hitro spreminjajoče se podatkovno središče.

Tako SDN-omrežja zagotavljajo:

a. Hitro in agilno orkestracijo omrežnih storitev
    - Kreiranje omrežnih elementov v programsko določenem omrežju mora biti enostavno kot kreiranje novega navideznega računalnika v navideznem okolju. Koncept programsko določenega omrežja je tako bolj primeren za navidezna okolja in tako posredno tudi za oblak.

b. Omrežno fleksibilnost in celovito upravljanje
    - Upravljanje programsko določenega omrežja ni omejeno s protokolom SNMP kot v klasičnem omrežju. Zato si lahko zagotovimo omrežno eksperimentiranje brez skrbi zaradi morebitnih posledic. Enostavno lahko testiramo določene nastavitve omrežja, preden jih apliciramo v realno omrežje.

c. Boljšo in granularno varnost
    - Zaradi arhitekture programsko določenega omrežja lahko zagotavljamo boljšo in natančnejšo varnost omrežnih naprav, aplikacije in s tem storitev.

d. Boljšo efektivnost in cenejšo operativnost
    - Investicija v programsko določeno omrežje je na začetku večja kot pri klasičnih računalniških omrežjih. Zato pa se predvideva, da se bo vzdrževanje omrežja drastično znižalo zaradi centralizacije njegovega upravljanja. V gospodarstvu so programsko določena omrežja kot prodajni model ovrednotena po metodologiji, ki temelji na izračunu privarčevanega denarja na dolgi rok.

e. Neodvisnost in navidezne omrežne storitve
    - Programsko določena omrežja imajo enak pomen za računalniška omrežja, kot ga ima strežniška virtualizacija za strežniško infrastrukturo. Strežniška virtualizacija je v zadnjem desetletju prispevala k znižanju stroškov strojne opreme in storitev v podatkovnih središčih. S tem se je povečala tudi kompleksnost upravljanja klasičnega vzdrževanja računalniškega omrežja. Klasično računalniško omrežje komunicira z viri

v omrežju preko IP-naslovov, kar ni zadostno za velika omrežja v podatkovnih središčih. Zato so potrebne dodatne lastnosti, ki jih zagotavljajo programsko določena omrežja. Take lastnosti tako dodatno omogočajo fleksibilnost atributov omrežja (prepustnost, zakasnitev ipd.) za različne podatkovne tokove, ki jih lahko vzdrževalci omrežja prilagajajo glede na potrebe. Podatkovna središča danes tečejo v mešanem (navideznem in fizičnem) okolju. Potrebna je rešitev, ki zagotavlja hitro povezljivost dinamičnih aplikacij in hitro organizacijo uskladitve zahtev uporabnikov. Programsko določena omrežja povečajo fleksibilnost in ohranjajo preglednost, upravljanje in avtomatizacijo celotnega omrežja.

3.2.1 Praktična vidik uporabe SDN

Kot vemo, v praksi v računalniškem omrežju ni idealnih pogojev, zato pri uporabi SDN-omrežij pridobimo tudi negativne učinke, ki za seboj potegne veliko različnih problemov, kot so:

a) Pomanjkanje strokovnjakov
   - Ker so tehnologije, ki se uporabljajo pri SDN-omrežjih še zelo sveže, ne obstaja veliko strokovnjakov v gospodarstvu, ki bi obvladovali principe načrtovanja, postavitve in vzdrževanja takih omrežij. Omrežnim strokovnjakom klasične omrežne arhitekture primanjkuje ustrezno znanje, potrebno za obvladovanje SDN-omrežij. Zaradi tega bodo morale podjetja in organizacije nameniti še veliko denarja za usposabljanje kadrov.

b) Nezrelost opreme in protokola
   - SDN-omrežja še niso dosegla končne rasti, zato se protokoli, ki se tu uporabljajo še vedno konstantno razvijajo. Posledica tega pa je premalo proizvajalcev omrežne opreme, ki bi adaptirali tehnologijo SDN-omrežij v svoje produkte. S tega naslova so cene omrežnih naprav SDN še vedno visoke v primerjavi s klasičnimi omrežnimi napravami in tako redko prisotne v trenutnem gospodarstvu.

c) Zmogljivost naprav
   - Zaradi dodane kompleksnosti moramo za delovanje SDN-omrežja zagotoviti dodatne računalniške vire, ki zagotavljajo kakovostno neprekinjeno delovanje omrežnih storitev. Klasične omrežne naprave tega ne zagotavljajo, zato potrebujemo namenske omrežne naprave in strežnike, ki omogočajo ustrezne računalniške vire. Tako dodatno povečamo stroške omrežja.

d) Kompleksnost omrežja
   - Ker se v SDN-omrežjih kontrolni in podatkovni nivo omrežja ločita, je tako potrebno uvesti nove omrežne protokole, ki zagotavljajo mednivojsko komunikacijo. Kontrolni nivo postane dodaten nivo abstrakcije v omrežju. S tem pa se nam kompleksnost omrežja še dodatno poveča.

e) Težavnost razhroščevanja
   - SDN-omrežja zaradi svoje kompleksne večnivojske arhitekture lahko povzroča preglavice pri ugotavljanju pojavitve napake na omrežju. S tem pa povzroča negotovost kakovosti delovanja omrežnih storitev.

f) Pomankanje izkušenj
   - Zaradi majhnega števila postavitev SDN-omrežja v gospodarstvu je opisanih praktičnih izkušenj na tem področju manj. Posledično je tudi zelo malo internetnih virov ali literature, na katere bi se lahko obrnili v primeru pojavitev napake v SDN-omrežju.

3.3 Arhitektura SDN

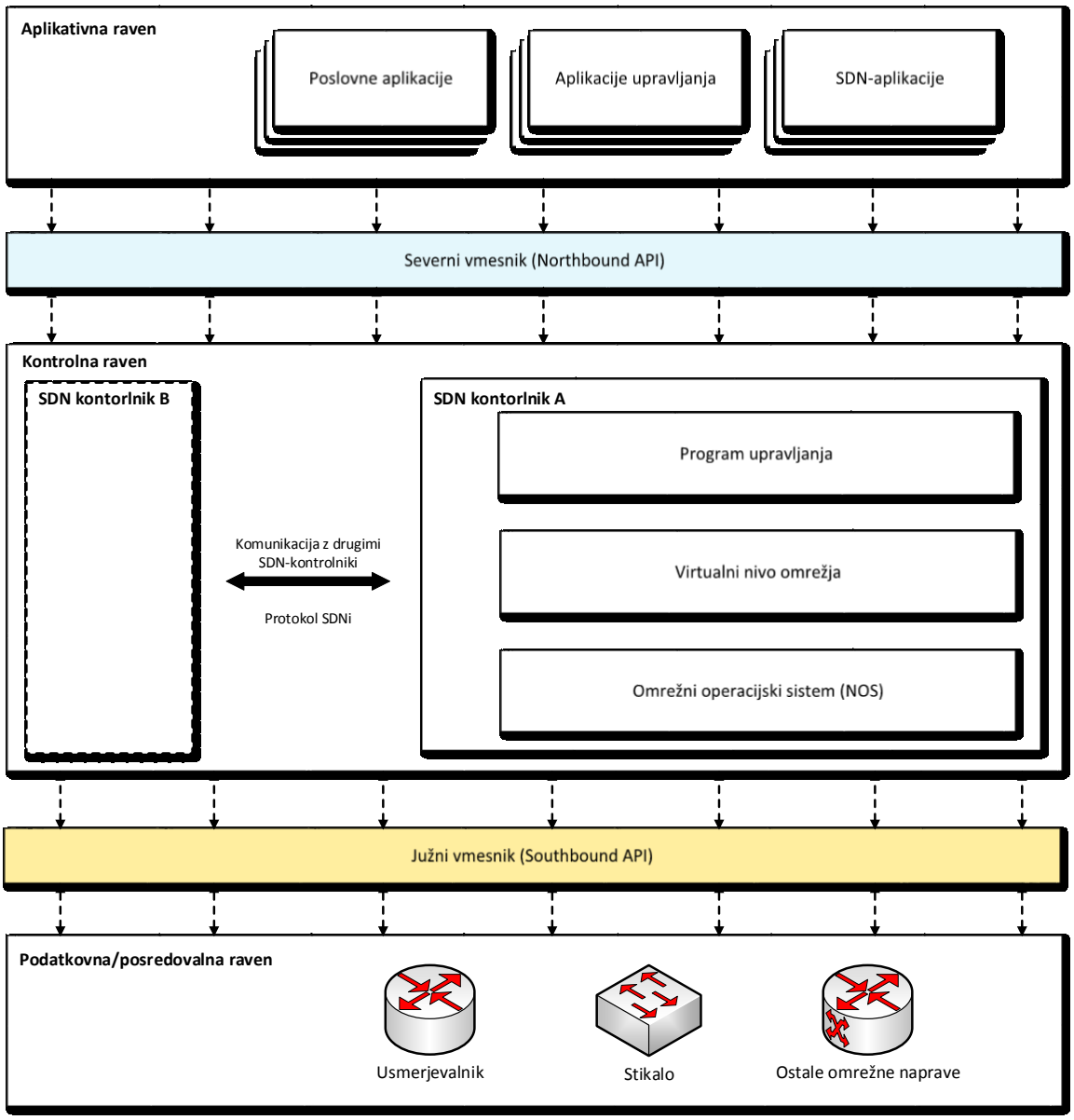
Arhitekturo programsko določenega omrežja lahko razdelimo na tri različne nivoje ali ravni (slika 3.3). Spodnja reven, ki ga imenujemo tudi podatkovna ali posredovalna raven, vsebuje omrežne naprave, ki zagotavljajo dejansko posredovanje podatkov (omrežnega prometa ali paketov). Nad tem nivojem je srednja raven ali tako imenovana kontrolna raven, ki vsebuje kontrolnik SDN. Komunikacija s kontrolnikom SDN med omrežnimi napravami na spodnji in srednji ravni poteka obojesmerno preko tako imenovanega južnega (southbound) vmesnika API. Najpogosteje uporabljen protokol, ki se ga uporablja za tako komunikacijo, je OF (OpenFlow) [16].

Najvišjo raven arhitekture programsko določenega omrežja predstavlja aplikacijska raven. Na tej ravni se nahajajo specifične omrežne aplikacije SDN, ki zagotavljajo omrežne storitve. Storitve zagotavljajo komunikacijo do kontrolnika SDN preko severnega (northbound) vmesnika API. Severni vmesnik ni točno standardiziran in je specifični del kontrolnika SDN. Za lažjo predstavo aplikacijske ravni si lahko predstavljamo primer storitve usmerjanja s protokolom BGP (usmerjevalni protokol, ki ima nalogo menjave poti in dosegljivosti informacij med avtonomnimi sistemi na internetu), ki teče kot storitveni proces na tej ravni (s tem simuliramo delovanje protokola BGP) [25].

Tudi kontrolnik SDN bi lahko predstavili kot večnivojski sistem, katerega spodnji nivo predstavlja omrežni operacijski sistem (NOS). NOS ima globalni pogled nad celotnim omrežjem in skrbi za komunikacijo do omrežnih naprav na podatkovni (posredovalni) ravni. Srednji ali navidezni nivo kontrolnika SDN zagotavlja abstraktni pogled na topologijo omrežja, najvišji nivo kontrolnika SDN pa omogoča implementacijo različnih specifičnih nalog ali storitev.

Na kontrolni ravni imamo lahko tudi več kontrolnikov SDN, ki tečejo v visokorazpoložljivem načinu. S tem lahko zagotovimo nemoteno delovanje programsko določenega omrežja tudi v primeru izpada enega ali več kontrolnikov SDN [9,10].

Z arhitekturo programsko določenega omrežja tako centraliziramo vodenje celotnega omrežja in zagotovimo dinamično prilagajanje omrežja storitvam na aplikativnem nivoju, hkrati pa lahko prispevamo k razširitvi funkcionalnosti oblačne arhitekture na nivoju infrastrukture kot storitev (IaaS).



Slika 3.3: Arhitekturne ravni programsko določenega omrežja

3.3.1 Protokol OpenFlow (OF)

Najbolj pogost in razširjen protokol za upravljanje z omrežnimi napravami v programsko določenem omrežju je protokol OF. Protokol OF je bil razvit na Univerzi Stanford in je v industriji kar hitro postal de facto standard. Nadaljnji razvoj in kontrolo nad protokolom je prevzela organizacija Open Networking Foundation (ONF). Kmalu so se razvoju pridružili številni vodilni proizvajalci omrežne opreme, ki so začeli proizvajati omrežne naprave s podporo protokolu OF [15].

Omrežne naprave, podprte z OF, uporabljajo tokovne tabele, ki delujejo v zaporednem načinu. To pomeni, da se pravila izvajajo v zaporednem vrstnem redu. Najprej se izvede prvo pravilo v seznamu tokovne tabele, rezultat tega pa je aktivacija drugega pravila. Tako se postopek nadaljuje do zadnjega pravila na seznamu tokovnih tabele.

Tokovna tabela je podatkovna struktura, ki se nahaja na podatkovni ravni SDN-omrežja v OF podprtih omrežnih napravah (npr. odprto omrežno stikalo ali stikalo OVS). Njena vsebina določa obnašanje paketov v OF podprtih omrežnih napravah. Tokovna tabela je sestavljena z ene ali več tokovnih entitet ali pravil (vrstic v tokovni tabeli, ki predstavljajo tokovna pravila), ki jih prikazuje tabela 1. Pravila tokovne tabele identificiramo preko primerjalnega in prioritetnega polja, ki skupaj točno določata pozicijo pravila v tokovni tabeli [16].

Tabela 1: Glavne komponente tokovnega pravila (vrstica) v tokovni tabeli

Primerjalno polje	**Prioritetno polje**	**Polje števec**	**Polje akcija**	**Časovne omejitve**	**Piškotki**

Vsako tokovno pravilo vsebuje:

- primerjalno polje, ki se uporablja za primerjanje z omrežnimi paketi,
- prioriteto polje, ki se uporablja za preferenco tokovnih pravil,
- polje števec, katerega naloga je štetje zadetkov paketa, ki ustreza posameznemu pravilu,
- polje akcija, ki izvede ustrezno akcijo nad paketom,
- polje s časovno omejitvijo, ki vsebuje maksimalno časovno vrednost veljavnosti tokovnega pravila,
- polje piškotki, ki uporablja kontrolnik SDN za filtriranje tokovne statistike (dodajanje, spreminjanje, brisanje pravil v tokovni tabeli).

Prihajajoče omrežne pakete tako pošljemo zaporedno skozi seznam tokovnih pravil, dokler ne naletimo na pravilo, pri katerem je vrednost v glavi paketa enaka primerjalnemu polju (glej tabelo 1. in 2) v tokovnem pravilu. Primerjalna polja v tokovnih pravilih se raztezajo (se nanašajo na podatke v glavah paketov nižjih plasti) med prvo in četrto plastjo, specificirane po referenčnem modelu OSI. Primerjalna polja tako zavzemajo vrednosti vhodnih vrat (Input Port), Ethernet naslov, IP-naslov, vrat TCP ali labelno (tag) vrednost (npr. Protokol MPLS). Polje lahko zavzema katerokoli vrednost. Ko najdemo ustrezno tokovno pravilo v tokovni tabeli, ki ustreza tej vrednosti, se proži akcija (npr. posreduj ali odvrzi paket).

Tabela 2: Potrebna določena primerjalna polja po protokolu OF

Primerjalna polja			
1. plast	2. plast	3. plast	4. plast
Vhodna vrata	Ethernet izvorni naslov	IPv4/IPv6 protokolna št.	TCP izvorna vrata
	Ethernet ponorni naslov	IPv4 izvorni naslov	TCP ponorna vrata
	Tip Etherneta	IPv4 ponorni naslov	UDP izvorna vrata
		IPv6 izvorni naslov	UDP ponorna vrata
		IPv6 ponorni naslov	

Tabela 3: Potrebno določene akcije po protokolu OF

Akcija	Opis
Izhod	Posreduje paket določenim vratom.
Odvrzi	Vsi paketi brez izhodnih akcij naj bodo odvrženi.
Skupina	Procesiraj paket preko določene skupine v skupinski tabeli.

Protokol OF definira tudi nekaj opcijskih akcij, kot so:

- Set-Queue, ki se uporablja pri vrstah za nastavljanje kakovosti storitev (QoS),
- Push-tag/Pop-tag, ki se uporablja pri VLAN-, MPLS- in PBB-glavah omrežnih paketov,
- Change-TTL, ki se uporablja pri spreminjanu vrednosti TTL za IP- in MPLS-protokol.

Omrežne naprave s podporo OF imajo poleg tokovnih tabel še dva tipa tabele, ki se imenujeta merilna (meter) tabela in skupinska tabela. Merilne tabele skrbijo za zagotavljanje kakovosti storitev QoS, kot so denimo velikosti čakalnih vrst paketov (rate-limiting). Entitete v merilnih tabelah temeljijo na treh komponentah:

- merilni identifikator, ki identificira unikatni merilnik,
- merilni pas, ki določa stopnjo pasu in način procesiranja paketa,
- števci, ki se povečujejo vsakič, ko se uporabi merilni pas.

Skupinske tabele se uporabljajo za izvedbo akcij na skupinskih tokov (group flows); primer kadar posredujemo pakete prvim aktivnim vratom v skupini (fail over), ali kadar jih posredujemo vsem vratom v skupini (multicast). Protokol OF uporablja številna primerjalna polja in akcijske tipe, ki so specificirani v dokumentaciji OF [16], ki ga je izdala organizacija ONF. Tabela 2 in 3 prikazujeta primerjalna polja in akcijske tipe, ki jih morajo podpirati omrežne naprave s podporo OF [16,17,18].

3.3.2 Južni vmesnik (Southbound API)

Kot je bilo že omenjeno, se južni vmesnik uporablja za komunikacijo med kontrolnikom SDN in z OF podprtimi omrežnimi napravami. Komunikacije med kontrolnikom in OF podprto omrežno napravo poteka v realnem času, kar pomeni, da se spremembe v pravilih kontrolnika takoj aplicirajo na omrežne naprave, podprte z OF. Protokol OF je prvi in tudi najbolj uporabljan protokol za komunikacijo na tej ravni. S pomočjo OF-protokola kontrolnik SDN po potrebi zahteve omrežja v realnem času spreminja in ureja tokovne tabele na z OF podprtih stikalih in usmerjevalnikih. Poleg protokola OF pa južni vmesnik uporablja še druge protokole, kot sta na primer OVSDB [19,20] ali NetConf [18].

3.3.3 Severni vmesnik (Norhtbound API)

Severni vmesnik je povezava med omrežnimi aplikacijami in kontrolnikom SDN. Aplikacije so lahko različnega značaja – poslovne, nadzorne ali upravljalne. Vmesnik omogoča, da lahko aplikacije komunicirajo s kontrolnikom SDN in tako vplivajo na delovanje samega omrežja. Kontrolnik SDN tako

lahko dobi povratne informacije od aplikacije med delovanjem in po potrebi prilagodi omrežje, da aplikaciji pomaga izboljšati izvajanje storitve.

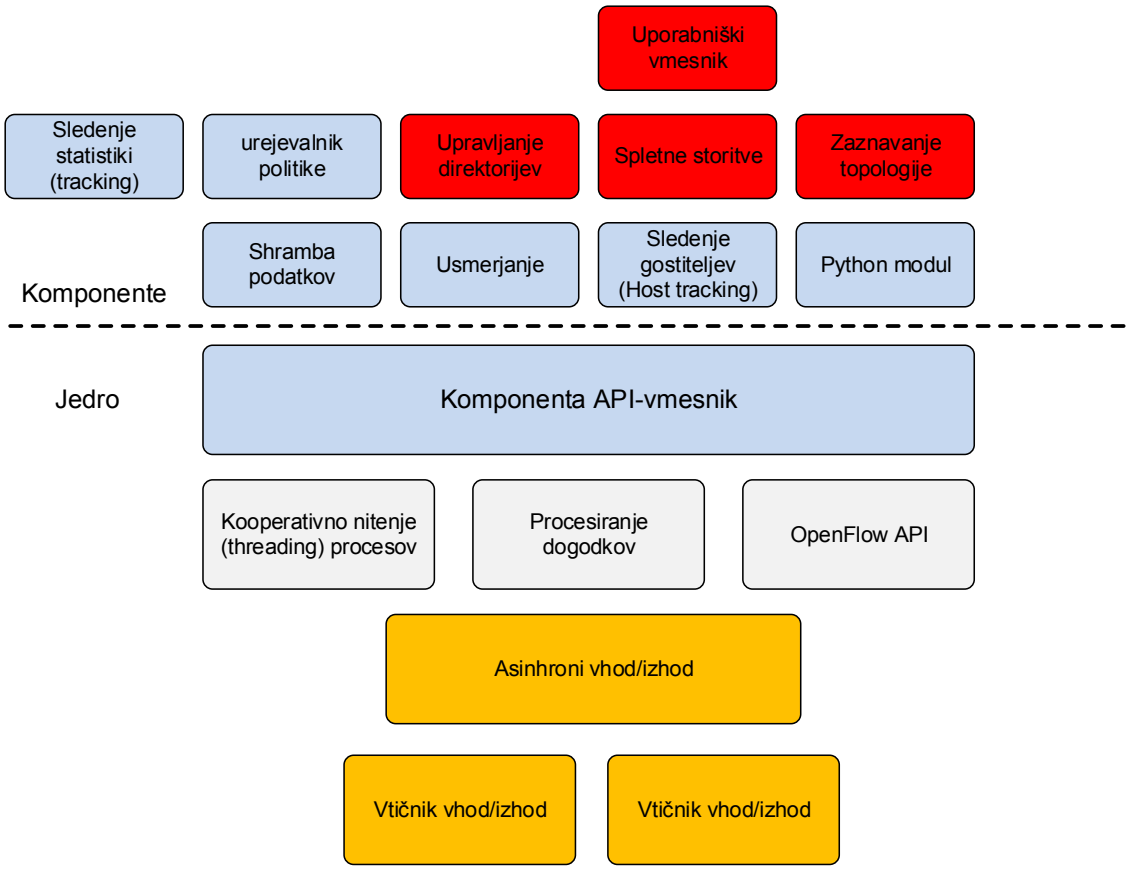
Kontrolnik SDN vedno komunicira s severnim vmesnikom preko spletnih storitev REST. Eden takih kontrolnikov je tudi kontrolnik ODL, ki ga bomo opisali kasneje v tem magistrskem delu. Severni vmesnik sicer ni standardiziran, se pa organizacija ONF zelo zavzema, da bi standardizirala vmesnik, tako kot je to naredila s protokolom OF. Če bi ONF to uspelo, bi vsaka SDN-aplikacija lahko komunicirala z vsemi kontrolniki SDN različnih proizvajalcev, podobno kot je to že dosegla z omrežnimi stikali, podprtimi z OF.

3.4 Kontrolniki SDN

Poznamo kar nekaj odprtih kontrolnikov (odprtokodnih), ki se uporabljajo v programsko določenih omrežjih. Ker je odprtokodnih kontrolnikov veliko, smo se osredotočili na štiri najpogosteje prisotne v odprtokodni skupnosti. Izbrani kontrolniki trenutno najbolj prispevajo k razvoju programsko določenih omrežij. Ti kontrolniki so POX, Ryu, Floodlight in OpenDaylight.

3.4.1 Kontrolnik NOX/POX

NOX in POX sta odprtokodni platformi za hitro razvijanje in izdelavo prototipov v programsko določenih omrežjih. Zastopana sta predvsem v akademski sferi. Razvita naj bi bila na Univerzi v Stanfordu [39], čeprav je težko pridobiti verodostojne vire njihovega nastanka. NOX je pisan v programskem jeziku C++ in je eden prvih kontrolnikov SDN, izdelanih za protokol OF [11]. NOX je komponento bazirano orodje za razvoj SDN-aplikacij. Podpira module, ki so specifični za protokol OF. Kontrolnik POX je novejša različica kontrolnika NOX in je pisana v visokonivojskem jeziku Python. Kontrolnik POX za razliko od svojega predhodnika omogoča preko vmesnika API višjenivojske poizvedbe, kot je poizvedba po topologiji omrežja, podpira pa tudi omrežno virtualizacijo.

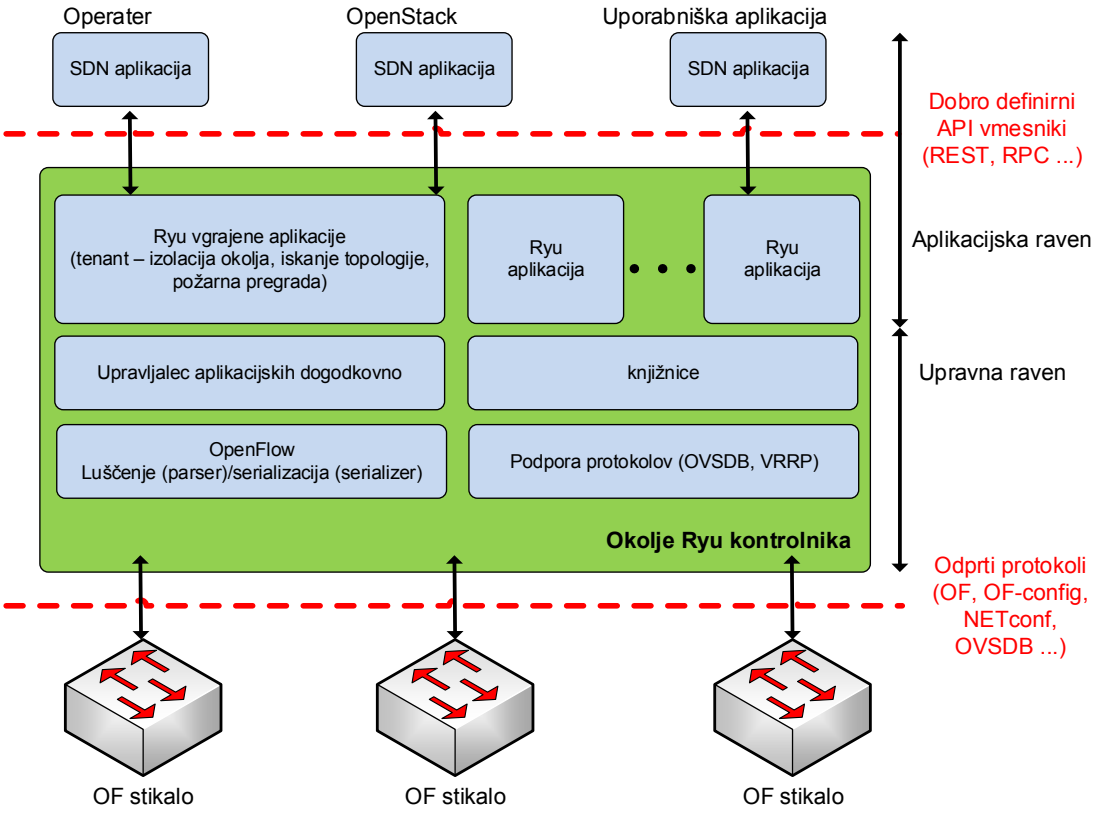


Slika 3.4: Prikaz arhitekture NOX/POX

Kontrolnika sta v konstantnem razvoju in sta v praksi neuporabna, saj imata kar nekaj hroščev. Slaba lastnost je tudi zastarel vmesnik API za protokol OF. NOX in POX sta omejena na protokol OF. Podpirata namreč samo različico 1.0 [21,22].

3.4.2 Kontrolnik Ryu

Odprtokodni kontrolnik Ryu je komponentno bazirano orodje, razvito v programskem jeziku Python. Sporočilna storitev v kontrolniku tako ne podpira komponente, pisane v drugih jezikih. Komponente kontrolnika vključujejo podporo za protokol OF, upravljanje dogodkov, sporočanje, aplikativno upravljanje, storitve infrastrukture in servis za ponovno uporabo (npr. knjižnice NETCONF, knjižnice sFlow / Netflow). Dodatno imamo na voljo funkcionalnosti, kot so sistem za zaznavanje napadov Snort (IDS), drugonivojsko stikalo po referenčnem modelu OSI, tunel GRE, topološke in statistične storitve omrežja. Kontrolnik Ryu vsebuje vtičnik za komponento oblačne platforme OpenStack Neutron (vmesnik API). Vtičnik podpira izdelavo tunela GRE kot navidezno privatno omrežje VLAN. Sam kontrolnik pa ima vmesnik REST za protokol OF [22,41].

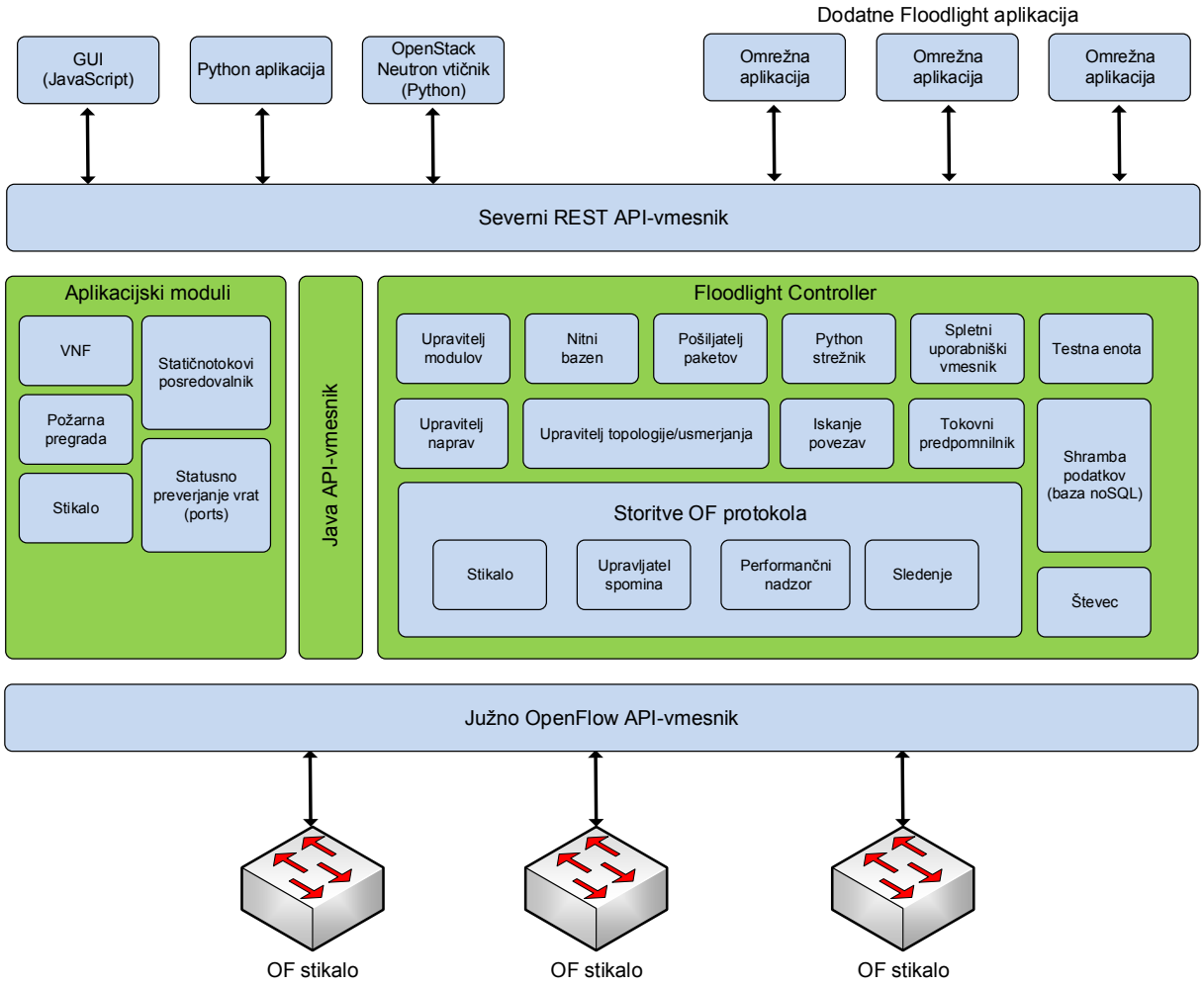


Slika 3.5: Arhitektura kontrolnika Ryu

Čeprav je dejansko zelo enostaven za implementacijo in uporabo, pa na žalost ne podpira načina visoke razpoložljivosti delovanja in je za velika omrežja zato neprimeren.

3.4.3 Kontrolnik Floodlight

Kontrolnik Floodlight [23] je razvilo podjetje Big Switch Networks po zgledu njegovega predhodnika Beacona [39], ki so ga razvili na Univerzi Stanford. Jedro arhitekture je modularno in vsebuje komponente za upravljanje topologije, naprav (sledenje naslovom MAC in IP), izračun poti, infrastrukture preko spleta, števce protokola OF in centralizirano abstraktno bazo (omogoča priklop SQL- in noSQL-baz). Komponente kontrolnika se vedejo kot storitve, ki podajajo stanja preko vmesnikov API. Te vmesnike kontrolnik uporablja za komunikacijo v oblaku. Obveščevalni sistem je programska ali strojna infrastruktura, ki podpira pošiljanje in sprejemanje sporočil med porazdeljenimi sistemi (MOM). Naloga sistema MOM je, da zagotovi, da sporočila pošiljatelja gotovo pridejo do želenih prejemnikov. Kontrolni vmesniki API omogočajo aplikacijam, da delajo poizvedbe stanja kontrolnika, lahko pa se naročijo na dogodek, ki ga oglašuje kontrolnik preko krmilnika dogodkov Java (Java Event Listener), kot je prikazano na sliki 3.6 [23].



Slika 3.6: Arhitektura kontrolnika Floodlight

Jedro vsega je upravitelj modulov, ki prevaja vhodno-izhodna sporočila omrežnih stikal, podprtih s protokolom OF, v dogodke kontrolnika Floodlight. Kontrolnik vsebuje nitni bazen, ki omogoča razporeditev obremenitve kontrolnika po modulih. Topološki modul skrbi za iskanje OF in ne OF omrežnih naprav s pomočjo protokola LLDP. Za kontrolnik Floodlight že obstaja nekaj aplikacij, ki opravljajo funkcijo stikala (stikalo s podporo protokola OF, ki ga razvijalci omrežne opreme upravljajo za integracijo z njihovo omrežno opremo), aplikacija HUB in aplikacija za statično dodajanje tokovnih pravil protokola OF. Kontrolnik vsebuje vtičnik za komponento oblačne platforme OpenStack (vmesnik API). Kontrolnik podpira trenutno vse verzije protokola OF ter lahko komunicira s fizičnimi in navideznimi stikali, podprtimi z OF. Kontrolnik vsebuje tudi odprtokodno knjižnico Loxi, ki omogoča drugim proizvajalcem omrežne opreme enostavnejšo in boljšo kompatibilnost s kontrolnikom Floodlight [22,23].

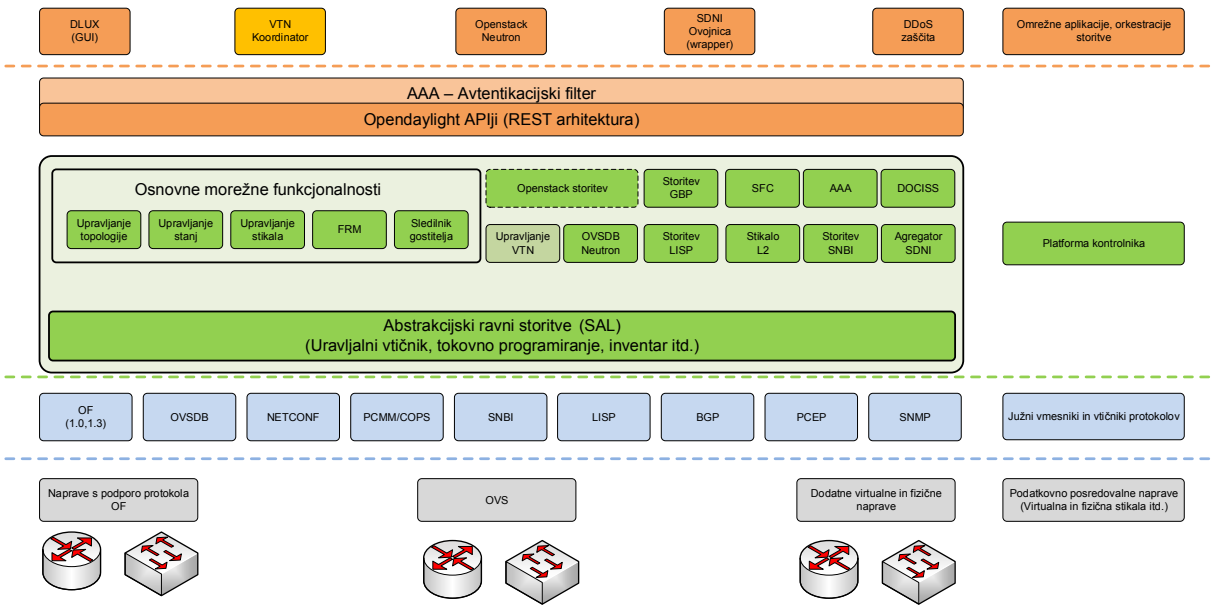
Kontrolnik Floodlight je izdan pod odprtokodno licenco Apache in je napisan v programskem jeziku Java. Ima bogat nabor orodij za razvoj in razhroščevanje kode kontrolnika. Uporablja spletno arhitekturo vmesnikov REST API in podatkovni model, ki zagotavlja konverzijo strukture YANG v REST. Protokol YANG API je spletni vmesnik RESTful API, ki razvijalcem spletnih storitev omogoča nadzor nad omrežnimi napravami. S tem omogoča enostavno objavo vmesnikov API in izmenjavo podatkov. Te funkcionalnosti se uporabljajo na različne načine, kot so prikazovanje, upravljanje, nadziranje različnih

podatkov v programsko določenih omrežjih (npr. prikaz topologije, upravljanje z različnimi stanji kontrolnika) [22].

3.4.4 Kontrolnik OpenDaylight (ODL)

Kontrolnik ODL je kontrolnik, ki ima trenutno največjo podporo v gospodarskem svetu in zelo dobro podporo tudi v odprtokodni skupnosti. Tudi sam sem imel priložnost sodelovanja v razpravah na ODL-forumu (ask.opendaylight.org). Obstaja pa tudi portal Wiki, ki je dobro dokumentiran in se konstantno dopolnjuje [24].

ODL je odprtokodni projekt, katerega cilj je razviti SDN-kontrolnik, ki bi služil kot osnova za razvoj programsko določenih omrežij. K razvoju kontrolnika ODL tako tehnično kot tudi finančno prispeva kar nekaj svetovno priznanih proizvajalcev programske in omrežne opreme. Najbolj znani so Cisco, IBM, HP, Brocade in RedHat. Arhitektura kontrolnika ODL je prikazana na sliki 3.7.



Slika 3.7: Arhitektura kontrolnika ODL

Ker je kontrolnik ODL napisan v programskem jeziku Java, teče v okolju JVM (Javanskem navideznem okolju, kar pomeni, da je ločen od ostalega okolja) in uporablja načela Open Services Gateway initiative (OSGi). [26]. Načela OSGi so nastala v odprti standardni organizaciji OSGi Alliance ter predpisujejo modularni sistem in servisno platformo za programski jezik Java, ki implementira dinamično komponentni model. Javanska orodja, razvita po teh načelih, imajo zelo dobro lastnost, da med delovanjem sistema omogočajo dodajanje, odstranjevanje in popravljanje modulov, ne da bi pri tem morali ponovno zagnati celotni sistem. ODL-moduli so vidni na sliki 3.7, označeni so z zeleno barvo in se nahajajo na abstrakcijski ravni storitve (SAL) na kontrolni ravni.

Kontrolnik ODL ponuja številne vtičnike, ki zagotavljajo vmesnike API in se razdelijo v dve skupini; to sta kontrolna skupina, ki se nahaja v platformi kontrolnika (glej sliko 3.7), in aplikativna skupina, ki

se nahaja na zgornjem nivoju kontrolnika ODL, kjer se nahajajo omrežne aplikacije in orkestracija storitev (glej sliko 3.7). Kontrolna skupina vsebuje notranje API-vmesnike in je namenjena za interno komunikacijo modulov. Aplikativna skupina vsebuje API-vmesnike, ki skrbijo za komunikacijo med moduli na platformi kontrolnika in omrežnim aplikacijami (zgornji nivo kontrolnika ODL) [25].

3.4.4.1 API-vmesniki na SAL

Naloga abstrakcijske ravni storitev (ki se nahaja v spodnjem nivoju platforme kontrolnika) je ločitev vmesnikov (API) aplikativne skupine od vmesnikov na platformi kontrolnika (južni moduli – OF, OVSDB, PCEP, BGP, SNMP itd.). Taka ločitev se je v praksi izkazala kot težka naloga. Tako je nastala abstrakcijska raven storitev SAL, ki je zbirka neodvisnih API-vmesnikov za vsak posamezni modul (namenjen posameznemu protokolu). Za ta namen so razvijalci kontrolnika ODL razvili dva tipa abstrakcijske ravni storitev (SAL):

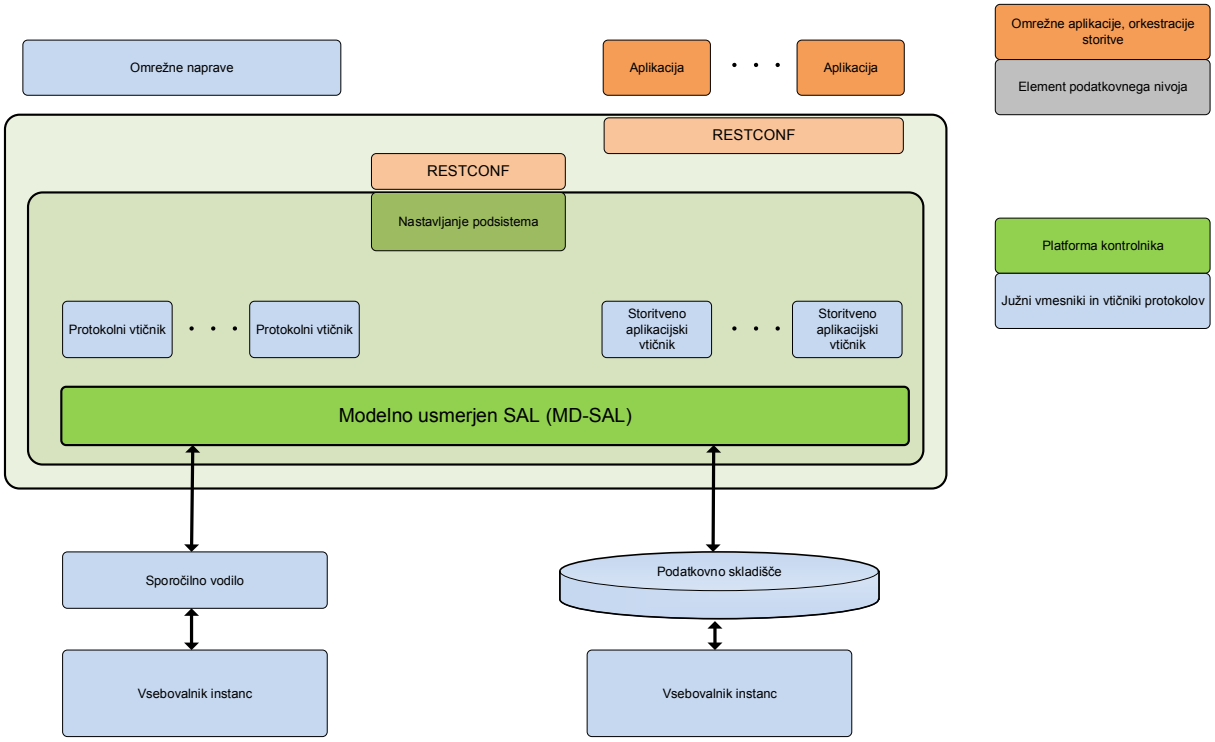
- AD-SAL, ki je abstrakcijska raven storitev, usmerjena na vmesnike API (AD – API-Driven), in skrbi za direktno komunikacijo z omrežnimi napravami, ki uporabljajo protokole, specifične za posamezne vmesnike API,

- MD-SAL, ki je modelno usmerjena (MD – Model-Driven) abstrakcijska raven storitev in skrbi za komunikacijo med modulnimi objekti, ki imajo nalogo, da na najbolj primeren način komunicirajo z omrežnimi napravami.

V AD-SAL vsakemu severnemu in južnemu vtičniku dodeli svoj vmesnik REST API. Pri MD-SAL pa se uporablja skupni vmesnik REST API za dostop do podatkov in funkcij v definiranih modelih. Tako MD-SAL združuje severne in južne vmesnike API in podatkovne strukture, ki se uporabljajo v različnih kontrolnikih SDN. Tako je način uporabe komunikacije po modelu MD-SAL bolj priljubljen, saj s tem najbolje ločimo in zaščitimo posamezne ravni (platformo in aplikacija) kontrolnika ODL. Vsaka aplikacija postane model, ki ima svoj vmesnik API, s katerim lahko komunicirajo druge aplikacije. S tem se tvori množica modelov (baza), v kateri so lahko vse aplikacije proizvajalci in potrošniki implementiranih storitev hkrati. Tako dobimo organizirano razporeditev odgovornosti in večnivojsko funkcionalnost [27,28]. S tem pristopom prispevamo tudi k boljši komunikaciji s kontrolniki drugih proizvajalcev.

3.4.4.2 MD-SAL

Moduli v kontrolniku ODL so med seboj povezani z vodilom MD-SAL, kot ga lahko vidimo na sliki 3.8. (sliko 3.8 je potrebno primerjati s sliko 3.7).



Slika 3.8: Modelno usmerjen SAL [29]

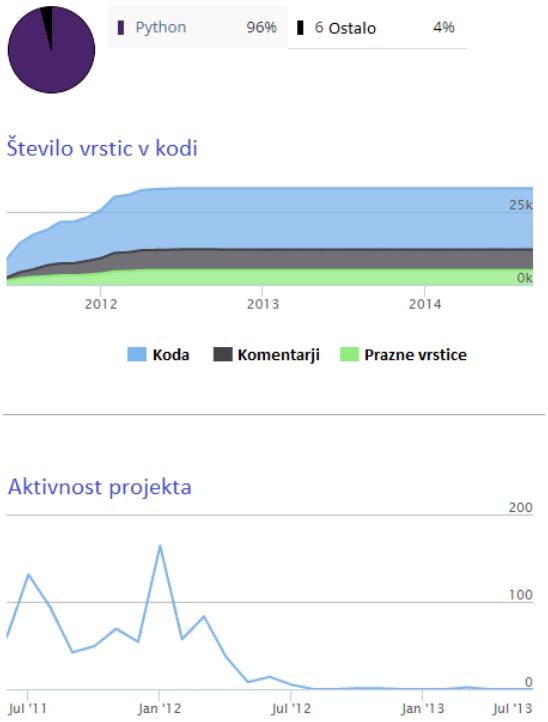
Z vidika razvijalca programske opreme med južnimi (southbound) in severnimi (northbound) vmesniki ni razlik. Pri razvoju aplikacije za kontrolnik je najbolje pripraviti ODL-model v okolju orodja OSGi [26]. Tako bo aplikacija za podatke v kontrolniku na voljo neposredno drugim modulom v načinu proizvajalec/potrošnik. Primer take aplikacije je učno (learning) stikalo, ki ga dobimo, ko namestimo kontrolnik ODL. To stikalo je zmožno oponašanja poznanih funkcionalnosti drugega nivoja po referenčnem modelu, kot sta ARP in STP. Ko pa učno stikalo prestavimo v kontrolno raven in ga povežemo z MD-SAL, pa lahko zelo enostavno sprejema vhodna sporočila (paket *packet_in*), ki prihajajo od omrežnih naprav do vtičnika za protokol OF. Lahko pa obratno preko vtičnika OF pošilja sporočila (*packet_out*) do omrežnih naprav. Sporočila enkapsulira, kar pomeni, da jim doda glavo v paketu na višjem nivoju po referenčnem modelu OSI. Ta funkcionalnost je zagotovljena preko vmesnikov REST API [28,29].

3.4.4.3 Varnost kontrolnika ODL

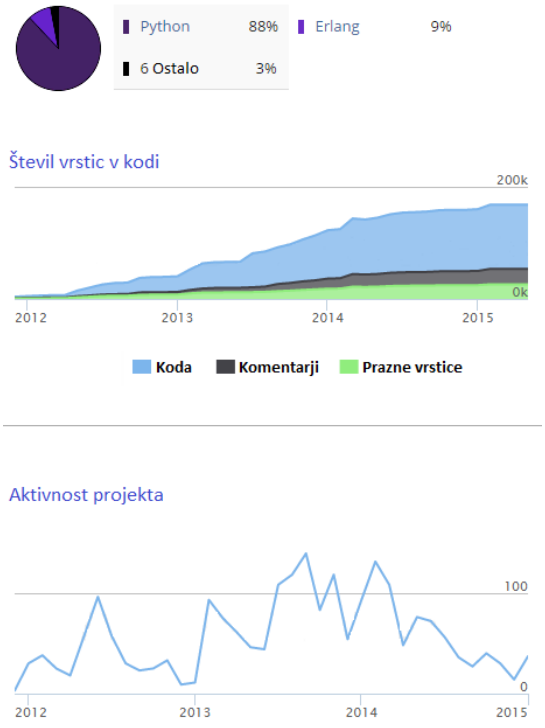
V času pisanja te naloge (junij 2015) je na trgu različica ODL Helium, ki je naslednik različice Hydrogen. Ker je Hydrogen prva različica kontrolnika ODL, se razvijalci niso kaj dosti ukvarjali z njegovo varnostjo, in je bila uporaba takih kontrolnikov SDN skoraj nepredstavljiva v produkcijskem okolju. Pri verziji Helium pa so razvijalci naredili kar nekaj korakov naprej. Tako imamo v novi različici mehanizem SNBI, ki zagotavlja avtentikacijo omrežnih naprav v kontrolniku ODL z izmenjavo ključev in z uporabo certifikatov IEEE 802.1AR. Mehanizem zagotavlja povezljivost preko protokola IPv6, ker vsaki napravi dodeli unikatni naslov. Nadzor dostopa poteka s pomočjo belih in črnih seznamov avtoriziranih naprav. Različica za avtentikacijo uporabnikov uporablja protokol AAA. Vsebuje tudi aplikacijo, ki preprečuje DDoS napade, ki prihajajo z omrežja [30].

3.4.5 Izbira kontrolnika SDN

Pri izbiri SDN-kontrolnika za programsko določeno omrežje smo upoštevali prisotnost posameznih kontrolnikov v gospodarstvu in interes proizvajalcev omrežne opreme pri njihovem razvoju. Pomemben kriterij je bil tudi, da so kontrolniki odprti. Za opisane kontrolnike smo pridobili (portalu Open Hub) še statistične podatke aktivnosti posameznega odprtokodnega projekta.



Slika 3.9: Prikaz aktivnosti projekta POX



Slika 3.10: Prikaz aktivnosti projekta Ryu

Slika 3.11: Prikaz aktivnosti projekta Floodlight

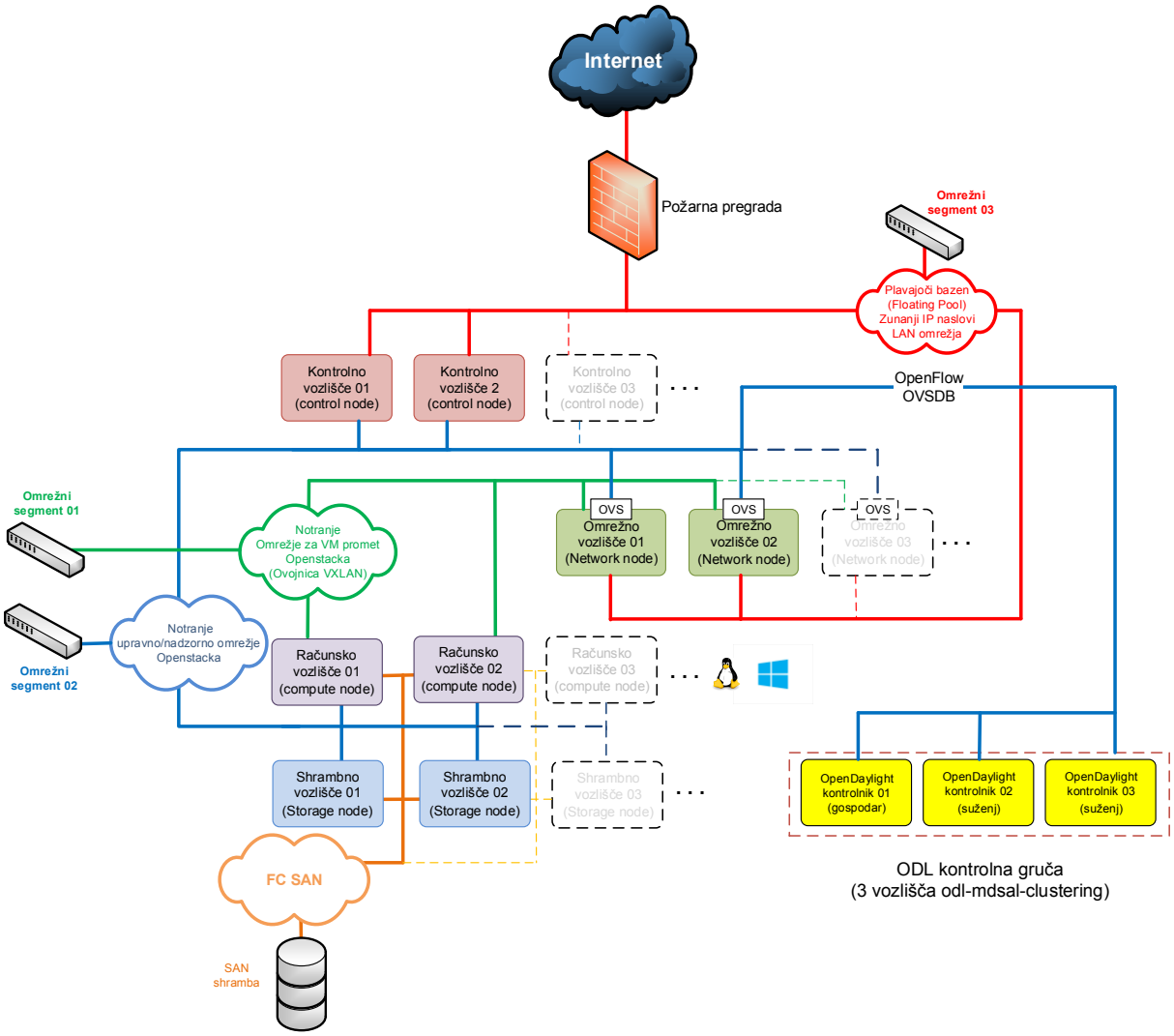
Slika 3.12: Prikaz aktivnosti projekta ODL

Slike 3.9, 3.10, 3.11 in 3.12 jasno prikazujejo aktivnost posameznega odprtokodnega projekta. Če primerjamo slike, lahko opazimo, da se tehtnica bolj nagiba v smer kontrolnika ODL. To se vidi predvsem v aktivnosti samega projekta ODL v zadnjem letu. Medtem ko sta POX in Ryu bolj prisotna v akademskih vodah, se kontrolnika Floodlight in ODL borita za tržni delež v gospodarstvu. Trenutno v odprtokodnem svetu prevladuje kontrolnik ODL, saj ima največjo podporo proizvajalcev programske opreme. ODL je tako postal najboljša izbira za integracijo programsko določenega omrežja v oblačno arhitekturo. Podatke s slik smo pridobili na spletnem portalu Open Hub (www.openhub.net).

4 Postavitev infrastrukture

Ker obstaja veliko različic odprtokodne oblačne platforme OpenStack, smo za namen priprave magistrskega dela izbrali različico OpenStack RDO (Really Darned Obvious ali Ridiculously Dedicated Openstackers). Projekt RDO je začelo podjetje RedHat in je najbolj priljubljena različica platforme OpenStack v Sloveniji (Ministrstvo za javno upravo) in tujini. Podjetje RedHat si je to platformo izbralo kot glavno oblačno platformo, okrog katere je zgradilo svoj finančni model poslovanja. Zaradi tega ima ta različica oblačne platforme OpenStack zagotovljen razvoj, ki se bo izvajal v naslednjih letih, in se ne bo tako hitro umaknila iz gospodarstva. RDO OpenStack teče na odprtokodnem derivatu operacijskega sistema RedHat Enterprise Linux, kot sta CentOS in Scientific Linux. Kot podporni operacijski sistem za implementacijo oblačne platforme pri magistrskem delu smo izbrali CentOS verzije 7 [31].

Na podlagi izkušenj in dobre prakse smo izdelali načrt za arhitekturo programsko določenega omrežja v oblaku. Arhitektura je prikazana na sliki 4.1. Ta načrt naj služi kot rezultat dela magistrske naloge in kot pravilen koncept postavitve take arhitekture v praksi [32].



Slika 4.1: Koncept postavitev oblačne platforme s programsko določenim omrežjem [32]

Na sliki 4.1 vsaka komponenta (vozlišče v obliki kvadrata) predstavlja svoj fizični (lahko tudi navidezni) strežnik, ki izvaja svojo oblačno storitev. Koncept arhitekture je zasnovan tako, da razdeli oblak v tri omrežne segmente, preko katerih tečejo različni tipi omrežnega prometa. Glavna segmenta sta upravno in nadzorno omrežje, kiskrbita, da vsa vozlišča v oblaku med seboj komunicirajo (omrežje modre barve). Kontrolno vozlišče in kontrolnik ODL uporabljata to omrežje za upravljanje celotne oblačne platforme in programsko določenega omrežja. Za komunikacijo med navideznimi računalniki v oblaku uporabljamo notranje VM-omrežje, preko katerega teče tunel VXLAN (omrežje zelene barve). Omrežje s plavajočim bazenom (omrežje rdeče barve) pa se uporablja kot komunikacijska pot notranjih oblačnih storitev z zunanjim svetom. V tem omrežju imamo bazen IP-naslovov, ki smo jih rezervirali v lokalnem fizičnem omrežju (LAN). Tega omrežja za potrebe magistrskega dela ne bomo uporabljali, saj ne bo potrebe po komunikaciji z zunanjim svetom, je pa to zelo pomembno pri postavitvi oblaka v produkcijskem okolju.

Arhitektura je zasnovana tudi tako, da omogoča dodajanje virov po širini oblaka (scale out). Arhitektura ima lastnosti, ki so potrebne za zagotavljanje visoke razpoložljivosti in s tem v primeru izpada posameznih oblačnih komponent (strežnikov) še vedno obdrži celotno funkcionalnost oblaka. To velja tudi za komponento kontrolnika ODL, ki nadzira programsko določeno omrežje. Kontrolnik ODL je namreč glavni element programsko določenega omrežja in tako največje ozko grlo pri implementaciji omrežja v oblaku. Zato je zelo priporočljivo, da se kontrolnik ODL postavi v gručnem načinu. Z gručnim načinom omogočimo, da se vsaka kontrolna zahteva omrežja porazdeli na različne kontrolnike ODL, ki skupaj delujejo kot en kontrolnik ODL. S tem razbremenimo posamezne kontrolnike v gruči in zagotovimo boljšo kakovost storitev (QoS) strežbe kontrolnih zahtev. To posledično prinaša visoko razpoložljivost, varnost in razširljivost kontrolnih podatkov programsko določenega omrežja v oblaku.

Pri sami implementaciji gručnega načina kontrolnikov ODL moramo strogo paziti, da gruča vsebuje minimalno tri vozlišča. Razlog za to je, da morajo biti topološki, inventarni in kontrolni podatki kontrolnikov ODL skupni, kar pomeni, da moramo v vsakem trenutku delovanja gruče imeti repliko teh podatkov. Pri implementaciji gruče z dvema vozliščema bi v primeru izpada enega od kontrolnikov (vozlišč) tako prišlo do izpada celotne gruče.

4.1 Postavitev oblačne platforme

Za potrebe magistrskega dela smo uporabili različico oblačne platforme OpenStack Juno, ki je v času izvedbe praktičnega dela (junij 2015) najbolj primerna. V razvoju je že nova različica, pod razvojnim imenom Kilo, vendar ima zaenkrat še nekaj težav s kontrolnikom ODL.

4.1.1 Namestitev OpenStack oblačne platforme RDO

Najprej je potrebno zaustaviti servis NetworkManage, ki skrbi za omrežne nastavitve na operacijskem sistemu, in ga onemogočiti, da se ne bo ponovno zagnal, in poskrbeti za izklop varnostega mehanizma v jedru operacijskega sistema (selinux):

```
$ systemctl stop NetworkManager
$ systemctl disable NetworkManager
$ systemctl enable network
$ sed -i 's/SELINUX=enforcing/SELINUX=disabled/g' /etc/selinux/config
```

Nato je potrebno nastaviti statične IP-naslove tako, da jih lahko omrežni servis pravilno uporablja pri vsakem vnovičnem zagonu OS.

Kreiramo datoteko `/etc/sysconfig/network-scripts/ifcfg-<interface_name>` in vpišemo potrebne podatke:

```
DEVICE=eth0
USERCTL=no
ONBOOT=yes
BOOTPROTO=none
IPADDR=<IP_strežnik1>
NETMASK=<omrežna_maska>
GATEWAY=<IP_naslov_prehoda>
```

Naslednja dva parametra prideta v poštev samo, če imamo v svojem okolju definirane imenske strežnike. V nasprotnem primeru uporabimo datoteko `/etc/hosts`.

```
DNS1=<imenski_strežnik1>
DNS1=<imenski_strežnik2>
```

Nato zaustavimo vse omrežne vmesnike:

```
$ ifdown <interface_name>
```

Zaženemo omrežni servis:

```
$ systemctl start network
```

Preden namestimo repozitorij RDO, preverimo, ali je potrebna nadgradnja operacijskega sistema. Ker za namestitev oblačne platforme uporabljamo orodje `yum`, nam ni treba skrbeti za odvisnosti posameznih programskih paketov, saj se bodo te same prenesle v repozitorij:

```
$ sudo yum update –y
```

Nastavitev repozitorij RDO, s katerega bomo namestili zadnjo verzijo OpenStack:

```
$ sudo yum install -y https://rdoproject.org/repos/rdo-release.rpm
```

Namestimo orodje `packstack` za nameščanje oblačne platforme OpenStack. `Packstack` je orodje, ki preko protokola SSH avtomatično namesti pakete in njihove programske odvisnosti (ostale pakete, ki so potrebni) s pomočjo programske opreme `Puppet` (program, ki omogoča oddaljeno upravljanje z računalnikom), ki so potrebni pri nameščanju komponent oblačne platforme OpenStack:

```
$ sudo yum install -y openstack-packstack
```

Sedaj sledi nameščanje platforme OpenStack. Najprej generiramo datoteko anwsers.cfg, ki služi kot vzorec za nameščanje oblačne platforme:

```
$ sudo packstack --gen-answer-file=<path-to-anwser-file>/answers.cfg
```

Nato popravimo datoteko `answers.cfg` tako, da ustreza želeni postavitvi oblačne platforme. Ker smo imeli na razpolago omejeno število virov (dva fizična strežnika), se je temu primerno prilagodilo celotno

oblačno okolje. Tako smo bili prisiljeni posamezne oblačne komponente združevati na en strežnik, kar je zadoščalo potrebam tega magistrskega dela. Posebno pozorni moramo biti na sledeče parametre:

```
# IP naslov strežnik, kjer bo nameščen kontrolnik
CONFIG_CONTROLLER_HOST=<IP_ali_FQDN_strežnika1>

# IP naslovi strežnikov, kjer bo nameščena komponenta Nova
CONFIG_COMPUTE_HOSTS=<IP_ali_FQDN_strežnika1>,<IP_ali_FQDN_strežnika2>

# IP naslovi strežnikov, kjer bo nameščena komponenta Neutron
CONFIG_NETWORK_HOSTS=<IP_ali_FQDN_strežnika1>,<IP_ali_FQDN_strežnika2>

# IP naslovi strežnikov, kjer bodo nameščena komponenta Glance in Cinder
CONFIG_STORAGE_HOST=<IP_ali_FQDN_strežnika1>,<IP_ali_FQDN_strežnika2>

# IP naslovi strežnikov, kjer bo nameščena komponenta AMQP, ki predstavlja #
sporočilni sistem in komunikacijsko vodilo za celoten oblak
CONFIG_AMQP_HOST=<IP_ali_FQDN_strežnika1>

# IP naslovi strežnika, na katerem bo podatkovna baza za kontrolno komponento
CONFIG_MARIADB_HOST=<IP_ali_FQDN_strežnika1>

# IP naslovi strežnika, na katerem bo podatkovna baza za komponento merjenja
# vrednosti oblaka ceilometer
CONFIG_MONGODB_HOST=<IP_ali_FQDN_strežnika1>

# Privatno ime omrežnega vmesnika na komponenti Nova, ki bo služilo za
storitev # DHCP in se nahaja v upravljavskem omrežju (plavo omrežje na sliki
4.1)
CONFIG_NOVA_COMPUTE_PRIVIF=XXX

# Javni omrežni vmesnik na komponenti Nova. Vmesnik, preko katerega bo tekel
# promet v zunanje omrežje (rdeče obarvano omrežje na sliki 4.1)
CONFIG_NOVA_NETWORK_PUBIF=YYY

# Privatni omrežni vmesnik na komponenti Neutron (strežnik Nova) (zeleno
# obarvano omrežje na sliki 4.1). Vmesnik, preko katerega bo tekel promet
# virtualnih strežnikov
CONFIG_NOVA_NETWORK_PRIVIF=ZZZ

# Nabor IP naslovov za upravljavsko omrežje (management network)(plavo
# obarvano omrežje na sliki 4.1).
CONFIG_NOVA_NETWORK_FIXEDRANGE=XXX.XXX.XX.X/XX

# Nabor IP naslovov za plavajoče omrežje (floating network) (plavo
# obarvano omrežje na sliki 4.1).
CONFIG_NOVA_NETWORK_FLOATRANGE=YYY.YYY.YY.Y/YY
```

Pomembno je nastaviti še parametre za ML2, ki omogoča komponenti Neutron uporabo tehnologij na drugem nivoju referenčnega modela OSI.

```
# Uporaba VXLAN-omrežja za povezovanje med fizičnimi strežniki (zeleno
# omrežje na sliki 4.1).
CONFIG_NEUTRON_ML2_TYPE_DRIVERS=vxlan
CONFIG_NEUTRON_ML2_TENANT_NETWORK_TYPES=vxlan

# Uporaba OVS (open vitrual switch), kot glavnega stikala za povezovanje
# med omrežnimi komponentami.
CONFIG_NEUTRON_ML2_MECHANISM_DRIVERS=openvswitch
```

Na koncu lahko popravljeno datoteko `anwser.cfg` uporabimo za namestitev platforme OpenStack s pomočjo orodja `packstack`:

```
$ sudo packstack --answer-file=<path-to-anwser-file>/answers.cfg
```

Vsebina celotne datoteke `answers.cfg` je pripeta v poglavju 7.1. Določeni parametri imajo prazne vrednosti, vendar jih kljub temu ne smemo odstranjevati z datoteke `answers.cfg`. Če želimo dodati novo računsko ali omrežno vozlišče, popravimo parametra `CONFIG_COMPUTE_HOSTS` in `CONFIG_NETWORK_HOSTS`, tako da dodamo k že obstoječim naslovom še IP- ali FQDN-naslov (celoten domenski naslov) novih vozlišč. Nato pa dodamo še parameter `EXCLUDE_SERVERS`, kateremu dodamo seznam IP-naslovov vozlišč, ki so že postavljeni (nameščeni) in jih želimo ob naslednji namestitveni iteraciji preskočiti. Primer je videti takole:

```
CONFIG_COMPUTE_HOSTS=<IP_ali_FQDN_strežnika1>,<IP_ali_FQDN_strežnika2>
CONFIG_NETWORK_HOSTS=<IP_ali_FQDN_strežnika1>,<IP_ali_FQDN_strežnika2>
EXCLUDE_SERVERS=<IP_ali_FQDN_strežnika1>
```

4.2 Postavitev programsko določenega omrežja

Pri postavitvi programsko določenega omrežja je potrebno razložiti, kako kontrolnik ODL integriramo v OVS-stikalo (Open Virtual Switch) na posameznih omrežnih vozliščih oblačne platforme OpenStack (omrežno vozlišče na sliki 4.1).

4.2.1 Namestitev infrastrukture na kontrolnem nivoju

Preden namestimo kontrolnik ODL, pripravimo ustrezno okolje. V operacijski sistem se prijavimo kot uporabnik `root`. V datoteko `.bashrc` zapišemo sledeče vrstice:

```
alias odl-client='/root/distribution-karaf-X.X.X-Helium-SRX /bin/client'
alias odl-stop='/root/distribution-karaf-X.X.X-Helium-SRX/bin/stop'
alias odl-start='/root/distribution-karaf-X.X.X-Helium-SRX/bin/start'
alias odl-log='tail -f /root/distribution-karaf-X.X.X-Helium-
SRX/data/log/karaf.log'
export M2_REPO=/root/.m2/repository
export JAVA_HOME=/usr/lib/jvm/jre-1.7.0-openjdk/
export MAVEN_OPTS="-Xms2048m -Xmx2048m -XX:MaxPermSize=2048m"
export MAVEN_OPTS="-Xmx1024m -XX:MaxPermSize=512m"
```

Sedaj lahko uporabljamo ukaze za zagon (`odl-start`), zaustavitev (`odl-stop`), vpogled dogodkov, ki se v živo dogajajo na kontrolniku ODL (`odl-log`) in vstop v konzolo kontrolnika ODL (`odl-client`). S tem smo na sistemu kreirali bližnjice, ki nam bodo olajšale delo pri samem sistemskem upravljanju kontrolnika ODL. Nato s spletne strani OpenDaylight namestimo programski paket, ki vsebuje verzijo kontrolnika ODL, ga odpakiramo in zaženemo.

```
$ cd /root/
$ wget https://<ODL naslov>/X.X.X-Helium-SR2/distribution-karaf-X.X.X-
Helium-SRX.tar.gz
$ tar xvfz distribution-karaf-X.X.X-Helium-SRX.tar.gz
```

```
$ cd distribution-karaf-X.X.X-Helium
$ odl-start
```

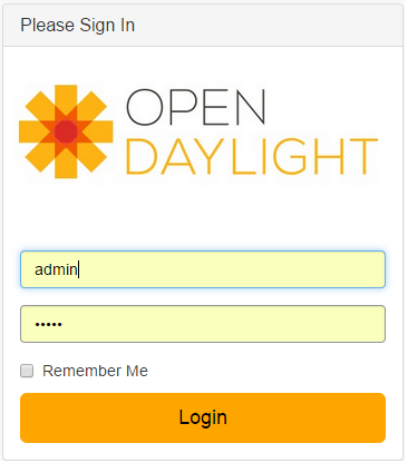
Po želji (v dobri praksi) lahko odpremo novo konzolo SSH in v njej zaženemo ukaz `odl-log`. Tako lahko opazujemo, kako se zaganja kontrolnik ODL, in ali med zaganjanjem ali med samim delovanjem ni prišlo do kakršnekoli napake. Kontrolnik ODL se konstantno razvija, zato se v dnevniku dogodka pogosto dogajajo napake, ki pa ne vplivajo toliko na osnovno delovanje kontrolnika. Določene napake, ki se pojavljajo v aktivnih dnevnikih, so pretežno del medsebojne komunikacije, neoptimiziranih modulov kontrolnika ODL. Ta informacija je bila pridobljena na ODL-forumu (ask.opendaylight.org).

Ko se kontrolnik zažene, se moramo vanj prijaviti in namestiti potrebne module, ki jih bomo potrebovali za nadaljnje delo. Prijavimo se na konzolo `Karaf`, ki je majhno okolje OSGi in jo kontrolnik ODL uporablja kot kontejner, na katerega lahko nameščamo različne komponente in aplikacije.

```
$ odl-client
opendaylight-user@root> feature:install odl-base-all odl-aaa-authn odl-restconf odl-nsf-all odl-adsal-northbound odl-mdsal-apidocs odl-ovsdb-openstack odl-ovsdb-northbound odl-dlux-core
```

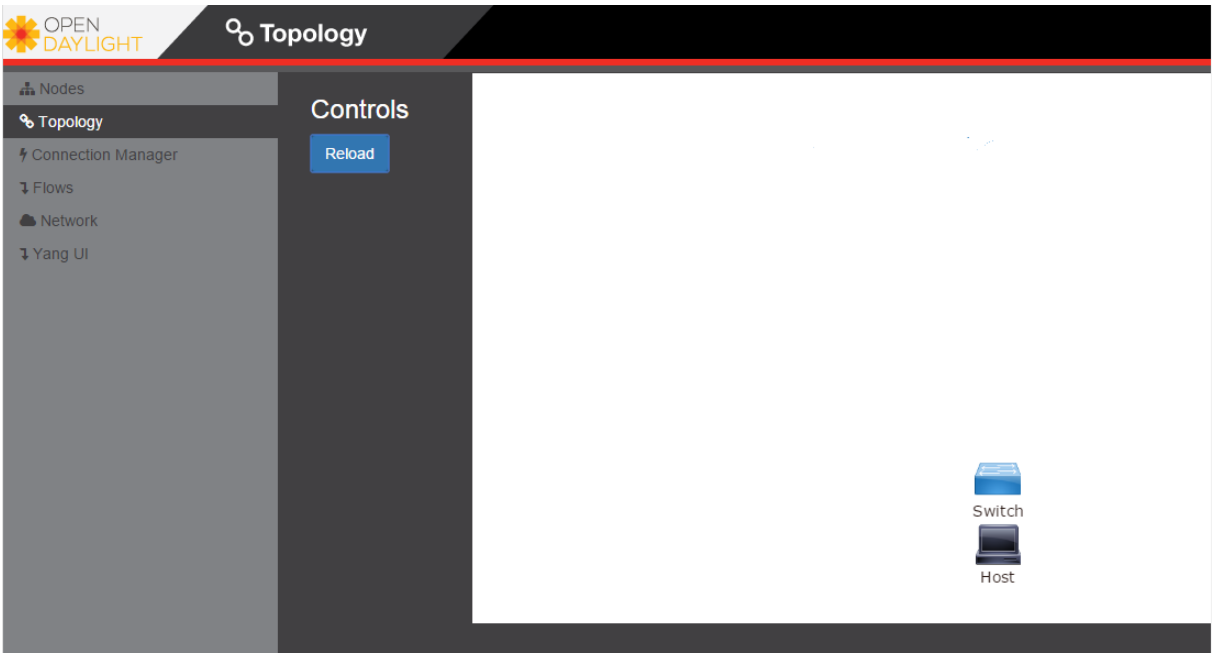
V magistrskem delu smo namestili kontrolnik ODL z različico Helium SR2. To tudi pomeni, da so ukazi in imena nameščenih modulov prilagojeni tej verziji in se lahko v nadaljnjih različicah kontrolnika spremenijo.

Po namestitvi modulov lahko hitro testiramo delovanje kontrolnika ODL tako, da se prijavimo v spletnem brskalniku na naslovu http://<IP_ODL_kontrolnika>:8181/dlux/index.html. Pokazati se nam mora spletno prijavno okno, kot ga vidimo na sliki 4.2.



Slika 4.2: Prijavno okno kontrolnika ODL

Ko se na spletu prijavimo v kontrolnik ODL, se nam prikaže grafični vmesnik (GUI), ki smo ga prej namestili z modulom `odl-dlux-core` in je prikazan na sliki 4.3.



Slika 4.3: Prikaz GUI-vmesnika dlux

Kot lahko vidimo na sliki 4.3, je vmesnik sestavljen iz menija na desni in prikazovalnega okna na levi strani. Spletni grafični vmesnik nam ponuja pet različnih pogledov, ki so na kratko opisani v tabeli 3.

Tabela 4: Različni pogledi v ODL spletnem grafičnem vmesniku dlux

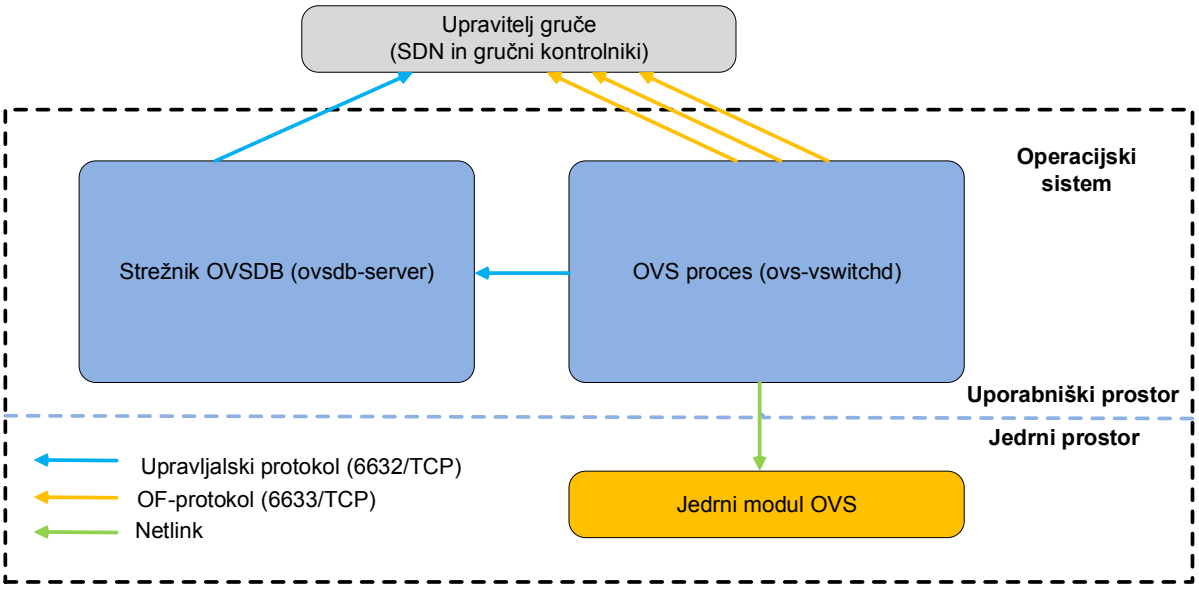
Pogled	**Opis**
Vozlišč	Pregled nad vsemi omrežnimi vozlišči (stikala, usmerjevalniki), ki so priklopljeni na kontrolnik ODL
Topologije	Pregled nad celotno topologijo omrežja
Tokov	Pregled podatkovnih tokov (flows), ki jih trenutno zazna kontrolnik ODL
Omrežja	Pregled omrežnih usmerjevalnikov v kontrolniku ODL
Yang vmesnik	Uporabniški vmesnik Yang, ki služi kot grafični prožilec RESTful akcij nad kontrolnikom ODL

4.2.2 Odprto navidezno stikalo (OVS)

V preteklosti so se strežniki v podatkovnem središču povezovali s klasičnimi fizičnimi stikali. S strežniško virtualizacijo pa se je povezovalna raven stikala spremenila v abstraktno navidezno raven. Stikalo OVS je odprta programska oprema, ki omogoča navidezno večnivojsko stikalo. Nameščeno je na gostitelju, ki gosti navidezne računalnike. Navidezni računalniki in kontejnerji (Docker) imajo logična ali navidezna mrežna vrata. Ta vrata se povezujejo na vrata stikala OVS.

Stikalo OVS so razvili v podjetju Nicira, ki ga je kasneje kupilo podjetje VMware. Ker v odprtokodni skupnosti ni boljše rešitve za navidezno stikalo, je stikalo OVS kmalu postalo de facto standard za navidezne odprtokodne platforme, kot so XEN, KVM in kasneje za oblačne platforme Eucalyptus, OpenNebula, CloudStack in OpenStack. Stikalo OVS podpira zelo veliko omrežnih protokolov, kot so

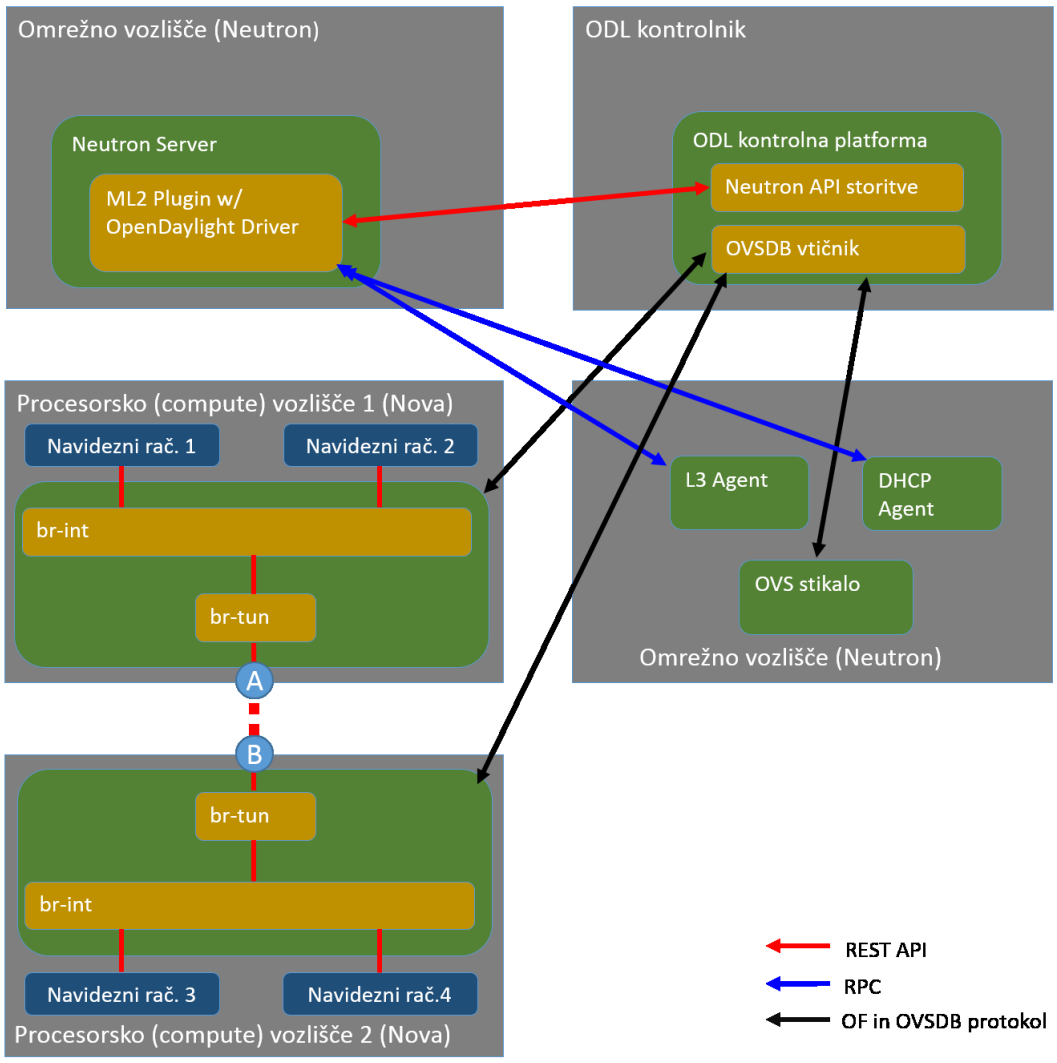
netflow, sflow, zrcaljenje vrat, VLAN, LACP, tunelski protokoli GRE, VXLAN, LISP in še mnogo drugih. Upravljanje stikala poteka preko protokola OVSDB in nadomešča star način upravljanja stikal s protokolom SNMP. Protokol OVSDB uporabljajo tudi nekateri proizvajalci omrežne opreme, kot so Cumulus, Arista in Dell. V programsko določenih omrežjih se za upravljanje stikala OVS uporabljata oba protokola OF in OVSDB. SDN-kontrolniki preko protokola OF upravljajo podatkovne tokove omrežnih paketov, s protokolom OVSDB pa upravljamo s samim OVS-stikalom. Pod upravljanje stikala OVS sodi kreiranje, nastavljanje, brisanje mostov (bridges), vrat (ports) in vmesnikov. Slika 4.5 prikazuje arhitekturo stikala OVS in njegove glavne komponente [34].



Slika 4.4: Arhitektura stikala OVS

Stikalni proces OVS skrbi za posredovanje logike učenja, zrcaljenja, vzpostavitev VLAN-omrežij ter oddaljeno konfiguracijo in vidnost omrežja. Strežnik OVSDB zagotavlja vmesnike RPC za komunikacijo do podatkovne baze OVSDB. Podatki potujejo preko JSON-RPC aktivne ali pasivne seje TCP/IP ali pa preko vtičnice (socket) Unix. Jedrni modul OVS je gonilnik v samem jedru operacijskega sistema in omogoča hitro pregledovanje, upravljanje in posredovanje omrežnih paketov, ki prihajajo in odhajajo na vmesnike stikala OVS. Skrbi tudi za enkapsulacijo in dekapsulacijo protokolnih ovojnic referenčnega modela OSI (dodajanje in odstranjevanje glav omrežnim paketom, ko gredo skozi referenčni model OSI). Upravitelj gruče skrbi za centralizirano gručno upravljanje in ni del stikala OVS. Tu gre predvsem za kontrolnike programsko določenega omrežja in gručne kontrolnike, ki se uporabljajo pri navideznih omrežjih (virtual networking) in imajo nalogo ločevanja različnih mostov (bridges) navideznih računalnikov od mostov drugih tipov prometa. Most v omrežni terminologiji pomeni združevanje oz. agregacijo dveh ali več ločenih omrežnih segmentov v eno omrežje. Razlikuje se od usmerjevalnika, ki omogoča komunikacijo med omrežnimi segmenti, ki ostajajo ločeni drug od drugega. Vsako stikalo OVS ima most, ki ima vrata z enakim imenom. Tako se notranji most za komunikacijo navideznih računalnikov imenuje `br-int` in vsebuje vrata po imenu `br-int`. Most, ki pa je povezan s transportnim nivojem hipervizorja, pa se imenuje `br-tun` in vsebuje vrata `br-tun`. Vsak paket, ki gre skozi logični vmesnik `br-int`, se bo razvrstil glede na pripadajoči tunel. Posredovanje paketov je funkcija gostitelja stikala OVS, zato to nalogo prevzemajo sistemska usmerjevalna pravila (routing tables). Izvorni IP-naslov je povezan z mostom `br-tun`, ki pa je

neposredno povezan s fizično mrežno kartico. Omrežni paket v tej točki prečka mejo navideznega omrežja v fizično omrežje. Prečkanje je prikazano na sliki 4.5.



Slika 4.5: Integracija stikala OVS s kontrolnikom ODL in oblačno platformo OpenStack

Slika 4.5 prikazuje integraciji stikala OVS s kontrolnikom ODL in oblačnim omrežnim vozliščem (Neutron) na platformi OpenStack.

4.2.3 Namestitev infrastrukture na podatkovnem nivoju (OVS in OpenStack Neutron)

Na podatkovnem nivoju smo uporabljali stikalo OVS, s katerim bomo upravljali preko protokola OVSDB s kontrolnikom ODL, kot je prikazano na sliki 4.5. Ker integracije kontrolnika ODL s platformo OpenStack ne moramo opraviti z orodjem `packstack`, moramo to storiti ročno.

Najprej moramo pripraviti okolje tako, da očistimo omrežne nastavitve, ki nam jih je orodje `packstack` namestilo v oblačni platformi. Spremeniti moramo delovanje omrežnega vozlišča oblačne platforme

OpenStack (Neutrona) tako, da nadomestimo modul ML2, ki v oblačni platformi skrbi za zagotavljanje drugonivojske omrežne storitve po referenčnem modelu OSI s kontrolnikom ODL. Nato moramo ustaviti in onemogočiti delovanje servisa `neutron-openvswitch-agent` na vseh strežnikih, kjer je nameščeno omrežno vozlišče oblačne platforme (Neutron). To storimo zato, da kontrolnik ODL zamenjamo z agentom L2 in ga upravljamo na daljavo s stikalom OVS.

Najprej odpremo mapo, kjer imamo avtentikacijsko datoteko (`keystonerc_admin`), ki se je generirala ob namestitvi oblačne platforme. Naslednji ukaz za odstranitev preostalih omrežnih nastavitev izvedemo v konzoli platforme OpenStack kot administrator:

```
$ . keystonerc_admin
$ neutron port-list
$ neutron port-delete id

$ neutron net-list
$ neutron dhcp-agent-list-hosting-net <ime_enega_z_net-list>
$ neutron dhcp-agent-network-remove <subnet UUID> <ime_enega_z_net-list>

$ neutron router-list
$ neutron router-port-list <ime_enega_z_route-list>
$ neutron router-interface-delete <ime_enega_z_route-list> <subnet_id>
$ neutron router-gateway-clear <ime_enega_z_route-list> <subnet_id>
$ neutron router-delete <ime_enega_z_route-list>

$ neutron subnet-list
$ neutron subnet-list id|name
$ neutron subnet-delete private-subnet

$ neutron net-list
$ neutron net-show private
$ neutron net-delete private

$ keystone tenant-list
$ keystone tenant-delete demo # če obstaja

$ neutron subnet-delete public-subnet
$ neutron net-delete public
```

Navedene ukaze izvedemo povsod, kjer se nahaja omrežno vozlišče oblačne platforme (glej sliko 4.1).

Kot smo že omenili, orodje `packstack` nima podpore za nastavitev kontrolnika ODL, zato bomo v naslednjih korakih onemogočili OVS-agenta in dodali podporo kontrolnika ODL, nato pa ponovno zagnali omrežno vozlišče oblačne platforme. Sledeči ukazi se izvajajo na konzoli kontrolnega vozlišča oblačne platforme (glej sliko 4.1):

```
$ sudo systemctl stop neutron-server
$ sudo systemctl stop neutron-openvswitch-agent
$ sudo systemctl disable neutron-openvswitch-agent
```

Sledi zaustavitev, čiščenje in ponovni zagon OVS-stikala in vseh dnevnikov (logs):

```
$ sudo systemctl stop openvswitch
$ sudo rm -rf /var/log/openvswitch/*
```

```
$ sudo rm -rf /etc/openvswitch/conf.db
$ sudo systemctl start openvswitch
```

Sledi nastavitev gonilnika za kontrolnik ODL :

```
$ sudo crudini --set /etc/neutron/plugins/ml2/ml2_conf.ini ml2 \
mechanism_drivers opendaylight
```

Sledi nastavitev tunela VXLAN:

```
$ sudo crudini --set /etc/neutron/plugins/ml2/ml2_conf.ini ml2 \
tenant_network_types vxlan
```

Sledi zapis avtentikacije in naslova API-vmesnika kontrolnika ODL:

```
$ cat <<EOT | sudo tee -a /etc/neutron/plugins/ml2/ml2_conf.ini > /dev/null
[ml2_odl]
username = admin
password = admin
url = http://<IP_naslov_ODL_kontrolnika>:8080/controller/nb/v2/neutron
EOT
```

Sledi čiščenje podatkovne baze na omrežnem vozlišču oblačne platforme:

```
$ sudo mysql -e "drop database if exists neutron_ml2;"
$ sudo mysql -e "create database neutron_ml2 character set utf8;"
$ sudo mysql -e "grant all on neutron_ml2.* to 'neutron'@'%';"
$ sudo neutron-db-manage --config-file /usr/share/neutron/neutron-dist.conf\
--config-file /etc/neutron/neutron.conf --config-file \
/etc/neutron/plugin.ini upgrade head
```

Nato izvedemo ponovni zagon procesa `neutron-server` na omrežnem vozlišču oblačne platforme:

```
$ sudo systemctl start neutron-server
```

Na kontrolnem vozlišču oblačne platforme zaženemo skripto clean_ovs.sh, ki je pripeta v poglavju 6.2, in očistimo vse preostale imenske prostore (namespaces), vrata (ports) in mostove (bridges), ki so še vedno ostali v oblačni platformi ob procesu nameščanja:

```
$ sudo ./clean_ovs.sh
$ sudo ovs-vsctl show
```

Če zadnji ukaz vrne prazno vrstico, potem omrežno vozlišče oblačne platforme nima nobenih nastavitev stikala OVS. Ko počistimo kontrolno vozlišče, moramo očistiti še vsa procesorska vozlišča (compute nodes) oblačne platforme (sliki 4.1):

```
$ sudo systemctl stop neutron-openvswitch-agent
$ sudo systemctl disable neutron-openvswitch-agent
```

Sledi ponovna zaustavitev, čiščenje in ponovni zagon stikala OVS in vseh dnevnikov (logs):

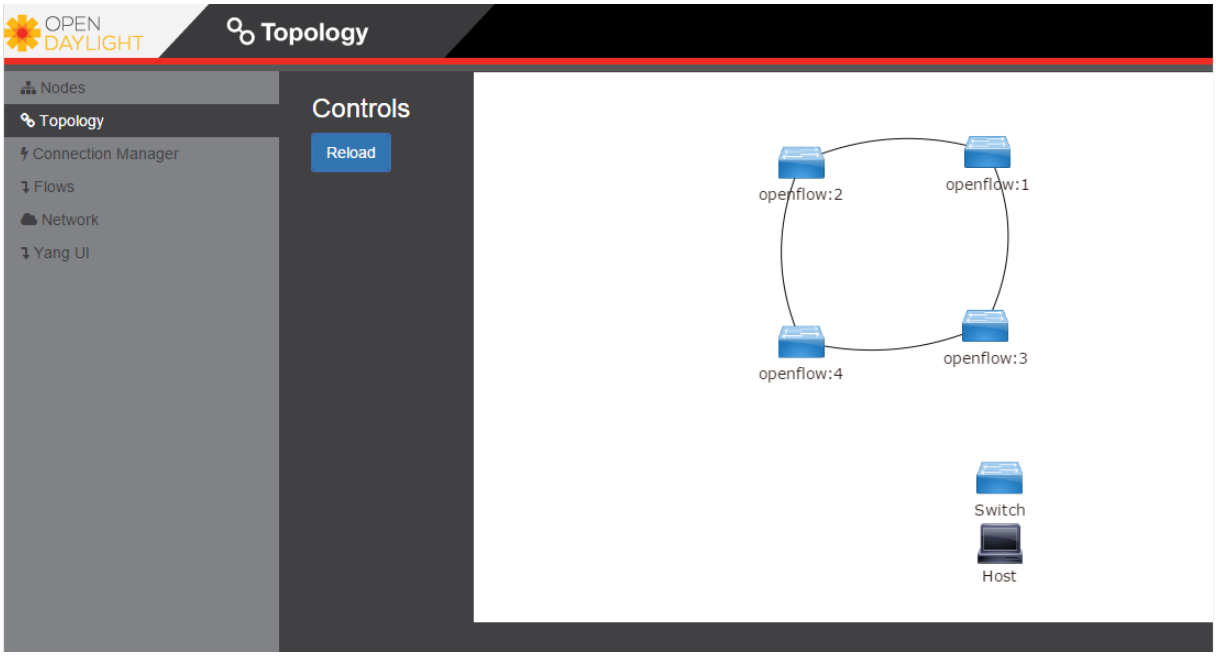
```
$ sudo systemctl stop openvswitch
$ sudo rm -rf /var/log/openvswitch/*
```

```
$ sudo rm -rf /etc/openvswitch/conf.db
$ sudo systemctl start openvswitch
$ sudo ovs-vsctl show
```

Nato prenesemo skripto `conf_ovs_opendaylight.sh` in jo zaženemo na vseh omrežnih vozliščih oblačne platforme, kjer imamo nameščeno stikalo OVS. Skripto si lahko pogledamo v razdelku 6.3. Zagon skripte poteka na naslednji način:

```
$./conf_ovs_opendaylight.sh <tunel-ip_naslov> <ODL_IP_naslov>
```

S skripto nastavimo vsa stikala OVS tako, da jih zazna kontrolnik ODL in jih lahko vidimo v topologiji na naslovu: http://<ODL_IP_naslov>:8181/dlux/index.html#/topology



Slika 4.6: Prepoznava OVS-stikal v kontrolniku ODL

5 Primer uporabe programsko določenega omrežja v oblaku

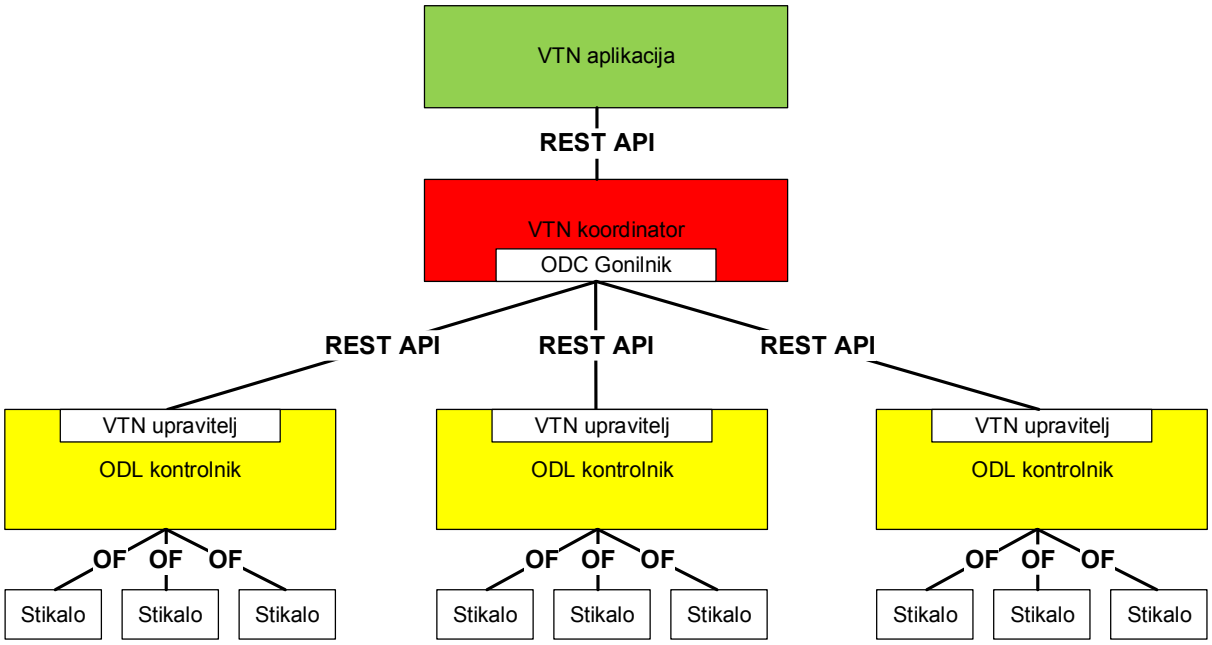
Veliko denarja je potrebnega pri postopku načrtovanja, implementacije ter nato nadziranja in upravljanja velikih in kompleksnih omrežij, zato so kakršnekoli kasnejše spremembe arhitekture na omrežju stroškovno in časovno zelo velika obremenitev. Ker je oblak dinamičen sistem, ki se prilagaja oblačnim storitvam, so arhitekturne spremembe omrežja še toliko bolj pomembne.

Zato smo s praktičnim primerom prikazali delovanje uporabe programsko določenega omrežja v oblaku. Za primer uporabe smo uporabili eno izmed že integriranih aplikacij kontrolnika ODL, ki se imenuje VTN (Virtual Tenant Network ali navidezno najemno omrežje). Z njim bomo pokazali, kako je mogoče spremeniti pravila obnašanja programsko določenega omrežja. Izvedli bomo poseg v arhitekturo omrežja.

Preden pa predstavimo primere uporabe, je potrebo podrobneje razložiti delovanje VTN-aplikacije v kontrolniku ODL.

5.1 Aplikacija VTN v kontrolniku ODL

Navidezno najemno omrežje (VTN) je aplikacija, ki zagotavlja kreiranje več najemnih navideznih omrežij (multitenant virtual network) v kontrolniku ODL in s tem omogoča hitrejše prilagajanje arhitekture omrežja. Enoličnost aplikacije VTN je njegova logična abstraktna raven, ki omogoča popolno ločitev od fizične ravni omrežja. Aplikacija VTN omogoča grajenje več različnih navideznih omrežij, ki so med seboj popolnoma ločena. Uporabniki kontrolnika ODL s tem lahko načrtujejo in implementirajo poljubna omrežja, ne da bi se zavedali topologije fizične ravni omrežja . VTN-aplikacija zagotavlja definiranje klasičnih omrežnih elementov na drugem in tretjem nivoju po referenčnem modelu OSI. Po končanem načrtovanju omrežja se načrtovana arhitektura omrežja avtomatsko namesti v spodnjenivojsko fizično omrežje. Distribucija poteka na vseh omrežnih napravah, ki podpirajo protokol OF in jih upravlja kontrolnik ODL (v magistrski nalogi bodo to stikala OVS). Novo uvedena logična raven, ki jo zagotavlja aplikacija VTN, tako skrije kompleksnost spodnjega omrežnega nivoja in poskrbi za boljše upravljanje omrežnih virov. S tem pridobimo veliko časa pri samem nastavljanju omrežnih storitev. Bolj pomembno pa je poudariti zmanjševanje pojava napake pri samem postopku nastavljanja celotnega omrežja [36].



Slika 5.1: Arhitektura VTN-aplikacije

Okolje VTN lahko obravnavamo kot neko veliko omrežno stikalo, pri katerem vsa vrata podpirajo protokol OF na spodnjem nivojskem omrežju.

Drevesna hierarhija na sliki 5.1 prikazuje arhitekturo aplikacije VTN. Njena glavna dela sta VTN-koordinator in VTN-upravitelj:

- VTN-upravitelj (manager) je vtičnik kontrolnika ODL, ki komunicira z drugimi moduli in s tem implementira omrežne komponente logične ravni modela VTN. Ponuja API-vmesnik, preko katerega VTN-koordinator proži zahteve REST ter s tem implementira in upravlja komponente VTN v kontrolniku ODL. Odgovoren je za upravljanje z vhodno-izhodnimi paketi protokola OF (`PACKET_IN` in `PACKET_OUT`), ki se vršijo na spodnjem omrežnem nivoju. Tako ima nalogo spreminjanja entitet tokovnih pravil OF (flow rules) na samem stikalu OVS s podporo protokola OF. Kot lahko vidimo na sliki 3.7 v poglavju 3, se VTN-upravitelj nahaja v sami platformi kontrolnika (označen s svetlo zeleno barvo). Na voljo je kot paket AD-SAL za kontrolnik ODL.

- VTN-koordinator je zunanja aplikacija, ki zagotavlja vmesnik RESTful API, preko katerega komunicirajo severne aplikacije (slika 3.7 v tretjem poglavju označen s svetlo oranžno barvo), ki jih zunanji uporabniki uporabljajo za izvajanje VTN-storitve. VTN-koordinator komunicira preko kontrolnega gonilnika ODL z vtičnikom VTN-upravitelja in s tem prenaša uporabniške nastavitve na elemente spodnjega omrežnega nivoja (npr. OVS-stikalo). Funkcionalnost VTN-koordinatorja se pokaže pri večjih omrežjih, kjer za nadzor in upravljanje omrežja ne zadostuje samo en kontrolnik ODL, temveč je potrebnih več. VTN-koordinator v tem primeru služi kot komunikacijski vmesnik z drugimi kontrolniki ODL, ki skupaj tvorijo federacijo kontrolnikov. Tako lahko tvorimo navidezno usmerjevalno vozlišče (vRouter), ki zna usmerjati omrežni promet na tretjem nivoju po referenčnem modelu OSI.

Za zagotavljanje navideznih omrežnih funkcionalnosti komponent VTN na logični ravni moramo najprej definirati okolje VTN. Okolje VTN nato povežemo s spodnjim omrežnim nivojem, kar omogoči interakcijo s fizičnim omrežjem. Način definiranja pravil v okolju VTN daje funkcionalnost filtriranja in usmerjanja omrežnega prometa na drugem in tretjem nivoju po referenčnem modelu OSI. Tabela 4. predstavlja vse elemente, ki jih lahko kreiramo v VTN-okolju.

Tabela 5: Tabela elementov, ki jih lahko kreiramo v VTN-okolju

Ime elementa		**Opis**
Navidezna vozlišča	vBridge	Logična reprezentacija funkcionalnosti drugonivojskega stikala po referenčnem modelu OSI
	vRouter	Logična reprezentacija funkcionalnosti usmerjevalnika
	vTep	Logična reprezentacija TEP (tunel s končno točko)
	vTunnel	Logična reprezentacija omrežnega tunela
	vBypass	Logična reprezentacija povezljivosti med upravljanimi omrežji
	vTerminal	Logični element, na katerega lahko vežemo logične vmesnike za ustvarjanje logičnih povezav
Navidezni vmesnik	Interface	Logična reprezentacija vmesnika končne točke v navideznem vozlišču
Navidezna povezava	vLink	Logična reprezentacija prvonivojske povezave navideznega vmesnika po referenčnem modelu OSI

Navidezno omrežje v okolju VTN je sestavljeno iz navideznih vozlišč (vBridge, vRouter), navideznih vmesnikov in navideznih povezav. S kreiranjem navidezne povezave na navideznih vmesnikih v navideznih vozliščih je mogoče pridobiti funkcionalnosti omrežnih nivojev 2 in 3 po referenčnem modelu OSI. V okolju VTN obstajajo povezave med omrežnimi fizičnimi viri in zgrajenim navideznim omrežjem. Vsaka takšna povezava identificira poslan ali sprejet omrežni paket navideznega omrežja, ki pripada stikalu OF (stikalo s podporo protokola OF) ter njegovemu pripadajočemu vmesniku na stikalu OF. Okolje VTN pozna dve metodi povezovanja.

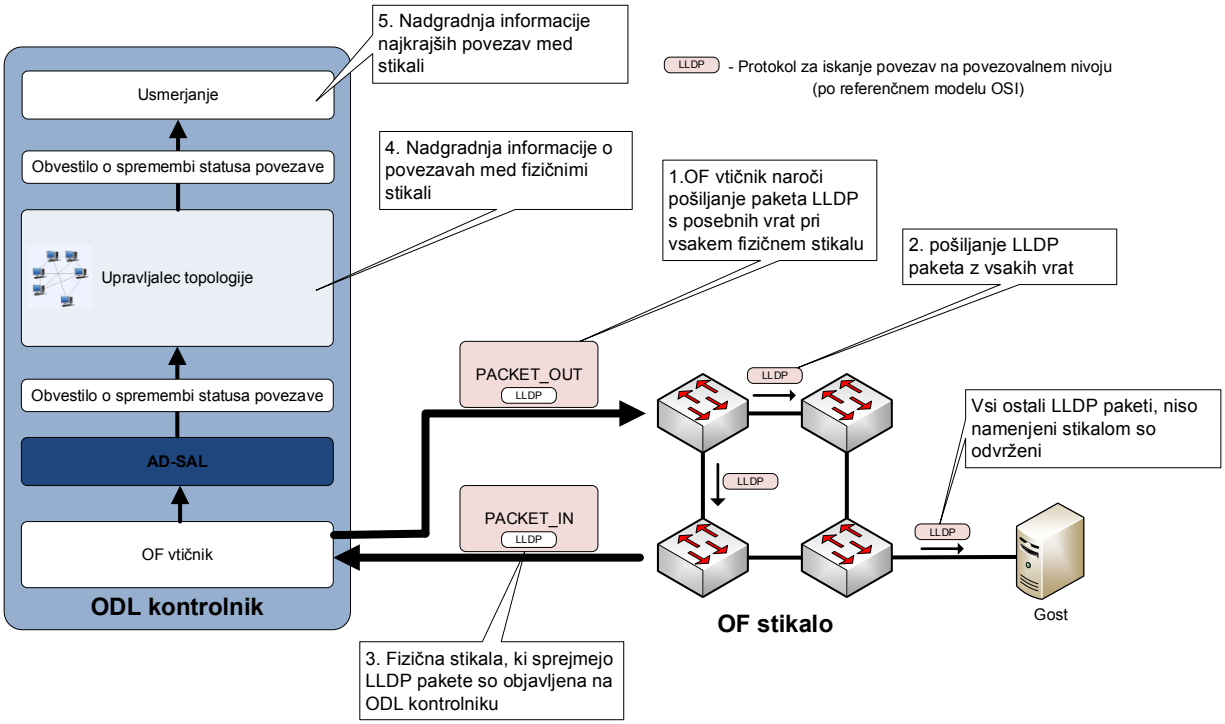
Prva metoda prispeli omrežni paket na stikalo OF poveže z vrati (port), ki ustrezajo povezovalni definiciji, nato se omrežni paket poveže z ustreznim navideznim vozliščem (vBridge) glede na ustrezno definirano povezavo VLAN.

Tabela 6: Tabela definicij povezav med navideznimi in fizičnimi omrežnimi elementi

Povezave	**Opis**
Povezava vrat	Povezovanje fizičnega omrežnega vira na vmesnik navideznega vozlišča (vBridge) s pomočjo ID stikala, ID vrat in VLAN ID podatkov prispelih paketov na drugem nivoju po referenčnem modelu OSI.
VLAN povezava	Povezovanje fizičnega omrežnega vira na navidezno vozlišče (vBridge) z uporabo podatka VLAN ID prispelih drugonivojskih paketov po referenčnem modelu OSI ali s pomočjo ID stikala in VLAN ID podatkov, prispelih paketov na drugem nivoju po referenčnem modelu OSI.

MAC povezava	Povezovanje fizičnega omrežnega vira na vmesnik navideznega vozlišča (vBridge) s pomočjo MAC naslova podatkov prispelih paketov na drugem nivoju referenčnega modela OSI.

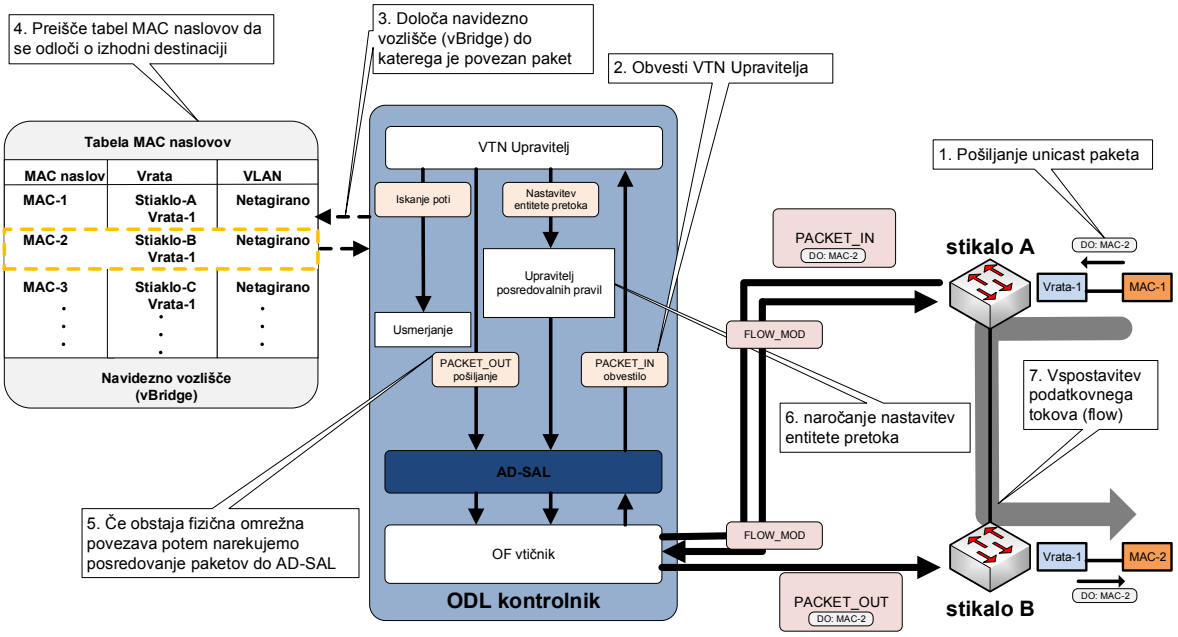
Po drugi metodi pa se lahko okolje VTN uči s pomočjo informacij, pridobljenih pri dostopanju do navideznega terminala (vTerminal), ki je povezan na stikalo OF. Informacije, ki se jih nauči z dostopom do navideznega terminala, so MAC naslovi in VLAN ID navideznega terminala v povezavi z vrati na stikalu. V primeru prekinitve povezave z navideznim terminalom se po določenem pretečenem času informacije o navideznem terminalu izgubijo (informacije zastarajo) [37].



Slika 5.2: Prikaz procesa dinamične izgradnje topologije SDN-omrežja

VTN-upravitelj si zgradi fizični pogled vseh povezav do stikal OF s pomočjo protokola LLDP. Omrežne naprave uporabljajo protokol LLDP za oglaševanje identitete in prepoznavanje njihovih sosednjih naprav. Postopek poteka tako, da se na vseh stikalih OF sproži pošiljanje paketov LLDP med stikali preko povezanih vrat. Stanje pošiljanja se beleži. S pomočjo podatkov o stanju pa se nato zgradi topološko omrežje, ki se konstantno nadgrajuje glede na naučene nove in spremenjene omrežne poti. Na

podlagi topologije se kreirajo tokovi pretokov omrežnih paketov (data flows). Proces dinamične izgradnje topologije omrežja je razviden s slike 5.2.



Slika 5.3: Proces ravnanja z novim paketom v omrežju

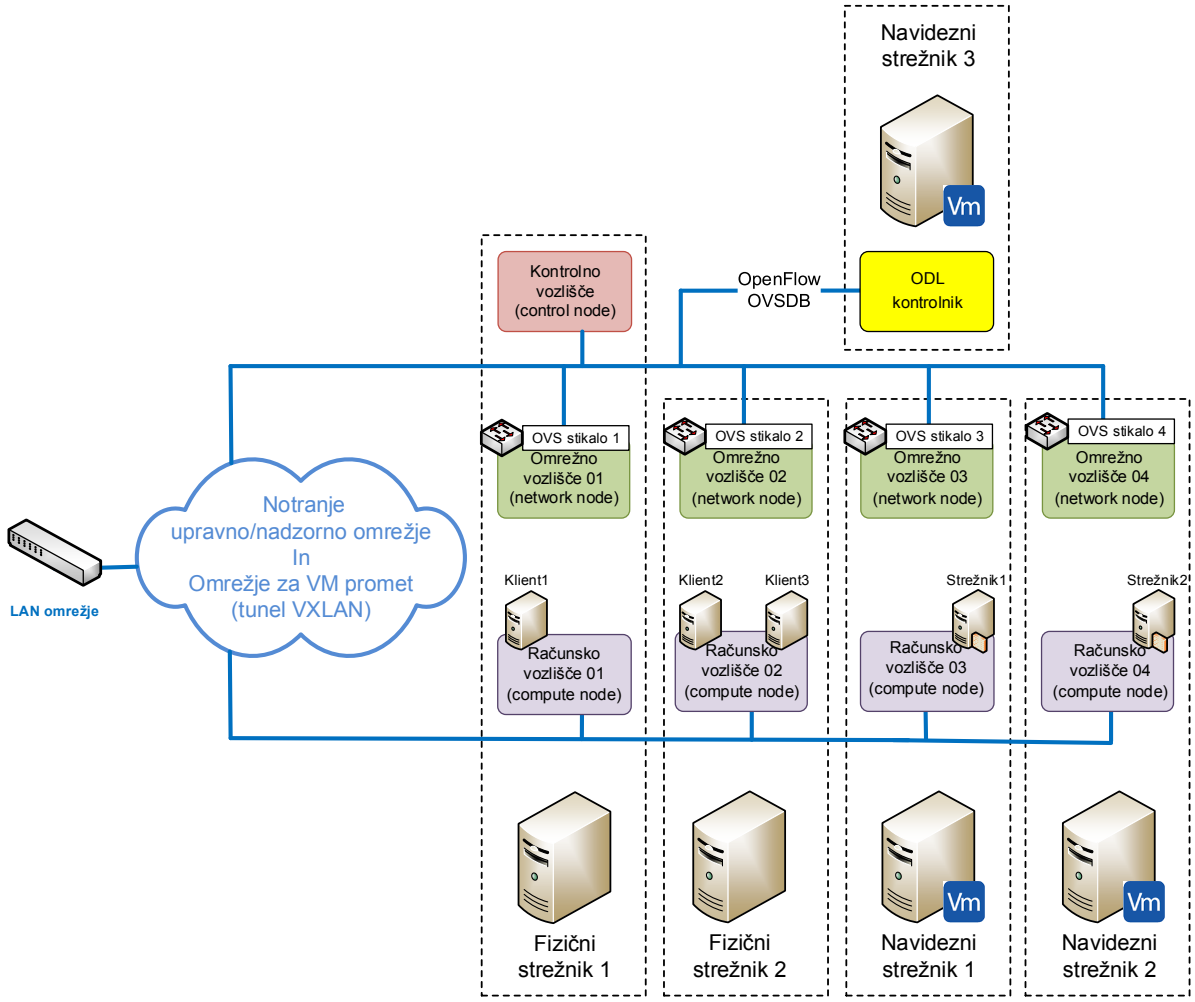
Slika 5.3. prikazuje postopke ravnanja z novoprispelimi omrežnimi paketi. Kot lahko vidimo, upravitelj VTN nemudoma preveri novoprispele pakete v tabeli MAC naslovov. Če naslov v njej ne obstaja, se sprožijo zahteve za pošiljanje paketov ARP skozi vmesnike v navideznem vozlišču (vBridge). Ko se tabela naslovov MAC posodobi, se sproži pošiljanje tokovnih entitet na končna vrata stikala OF. Te entitete vsebujejo informacijo o implementaciji podatkovnih tokov omrežja. Na podlagi topologije omrežja, pridobljene v prej opisanem postopku na sliki 5.2, lahko kreiramo ali popravimo podatkovne tokove tako, da ustrezajo naši topologiji omrežja. Tabela MAC naslovov se pasivno obnavlja glede na časovno omejene tokovne entitete ali do novega postopka pošiljanja paketov ARP (broadcast) [37].

5.2 Primer uporabe programsko določenega omrežja v oblaku

Primer uporabe delovanja programsko določenega omrežja bomo ponazorili s tremi primeri. S prvim primerom bomo prikazali pošiljanje omrežnih paketov po programsko določenem omrežju med dvema odjemalcema skozi omrežno vozlišče. Z drugim primerom bomo pokazali preusmeritev omrežnih paketov skozi dodatno omrežno vozlišče. S tretjim primerom pa bomo pokazali, kako lahko izoliramo odjemalca v omrežju in s tem zavarujemo omrežje pred okužbo z virusom. Vse omenjene primere uporabe bomo prikazali z uporabo aplikacije VTN na kontrolniku ODL, ki kreira dodatno navidezno raven za upravljanje podatkovnih tokov v navideznem omrežju preko protokola OF.

Za postavitev testnega okolja oblačne platforme je bilo na voljo premalo strojnih virov, da bi zgradili arhitekturo, prikazano na sliki 4.1 v poglavju 4. Zato je bilo potrebno združevati več oblačnih vozlišč (komponent) na enem fizičnem strežniku. Tako so se združile komponente omrežnega in procesorskega

vozlišča. Za primer prikaza uporabe programsko določenega omrežja smo uporabili dva fizična in tri navidezne strežnike, ki tečejo v ločenem navideznem okolju. Za lažjo predstavo podobe testnega okolja si lahko pogledamo sliko 5.4.

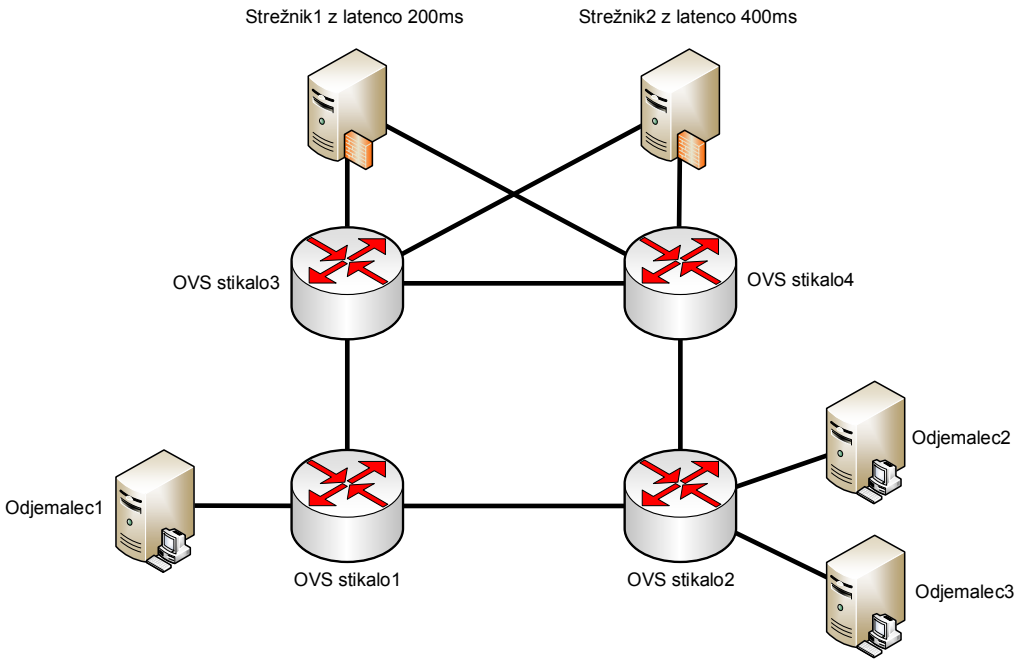


Slika 5.4: Pogled arhitekture testnega okolja

Kot lahko vidimo na sliki 5.4, smo je na prvem fizičnem strežniku postavili kontrolno, omrežno in računsko vozlišče, na drugem fizičnem strežniku pa samo omrežno in računsko volišče. Enako se zgodi na drugem fizičnem strežniku, kontrolnik ODL pa smo postavili na tretjem navideznem strežniku. Slika 5.4 prikazuje samo en omrežni segment, ki pa je uporabljen za dva tipa omrežnega prometa. Eden uporablja kontrolno vozlišče za upravljanje in nadzor nad vozlišči oblačne platforme, drugi tip prometa pa uporablja kontrolnik ODL za upravljanje stikal OVS preko protokola OF in OVSDB. Za omrežno infrastrukturo smo uporabili eno fizično omrežno stikalo. Vsak izmed fizičnih strežnikov ima samo eno omrežno kartico. Omenjene strojne omejitve moramo upoštevati pri izvajanju testov pošiljanja paketov po programsko določenem omrežju.

Predvsem je potrebno opozoriti, da je arhitektura testnega okolja strogo neprimerna za uporabo postavitve v produkcijskem okolju. Za postavitev v produkcijskem okolju moramo uporabljati arhitekturo, predstavljeno na sliki 4.1 v poglavju 4. Topologija navideznega omrežja testnega okolja je

prikazana na sliki 5.5. Topologija navideznega omrežja vsebuje štiri stikala, s katerimi so povezani trije odjemalci in dva strežnika.



Slika 5.5: Topologija omrežja testnega okolja

Za pripravo testne topologije s slike 5.5 moramo izvesti serijo ukazov v konzoli kontrolnega vozlišča oblačne platforme. Za navidezne strežnike smo uporabili sliko operacijskega sistema Linux različice CirrOS, ki je bila razvita za namene testiranja oblačnih platform in zavzema zelo malo strojnih virov. Sledeči ukazi v konzoli kreirajo vzorec slike CirrOS, ki jo lahko nadalje uporabljamo za kreiranje naših odjemalcev in strežnikov:

```
$ . keystonerc_admin
$ glance image-create --copy-from http://download.cirros-\
cloud.net/0.3.1/cirros-0.3.1-x86_64-disk.img \
  --is-public true \
  --container-format bare \
  --disk-format qcow2 \
  --name cirros
$ nova flavor-create cirros.nano auto 128 1 1
```

Sedaj izvedemo ukaze za kreiranje navideznih omrežnih elementov, kot so navidezno podomrežje in usmerjevalnik povezave do našega podomrežja 10.0.0.0/24:

```
$ neutron net-create vx-net --provider:network_type vxlan --\
provider:segmentation_id 1400
$ neutron subnet-create vx-net 10.0.0.0/24 --name vx-subnet
$ neutron router-create vx-router
$ neutron router-interface-add vx-rtr vx-subnet
```

Nato kreiramo navidezne odjemalce, ki jih priklopimo na navidezno podomrežje `vx-net`:

```
$ nova boot --flavor cirros.nano --image $(nova image-list | grep 'cirros'\
| awk '{print $2}' | tail -1) --nic net-id=$(neutron net-list | grep -w \
```

```
vx-net | awk '{print $2}') Odjemalec1 --availability_zone=nova:openstack01

$ nova boot --flavor cirros.nano --image $(nova image-list | grep 'cirros'\
| awk '{print $2}' | tail -1) --nic net-id=$(neutron net-list | grep –w \
vx-net | awk '{print $2}') Odjemalec2 --availability_zone=nova:openstack02

$ nova boot --flavor cirros.nano --image $(nova image-list | grep 'cirros'\
| awk '{print $2}' | tail -1) --nic net-id=$(neutron net-list | grep –w \
vx-net | awk '{print $2}') Odjemalec3 --availability_zone=nova:openstack02
```

Sledi kreiranje navideznih strežnikov, ki jih prav tako priklopimo na isto omrežje:

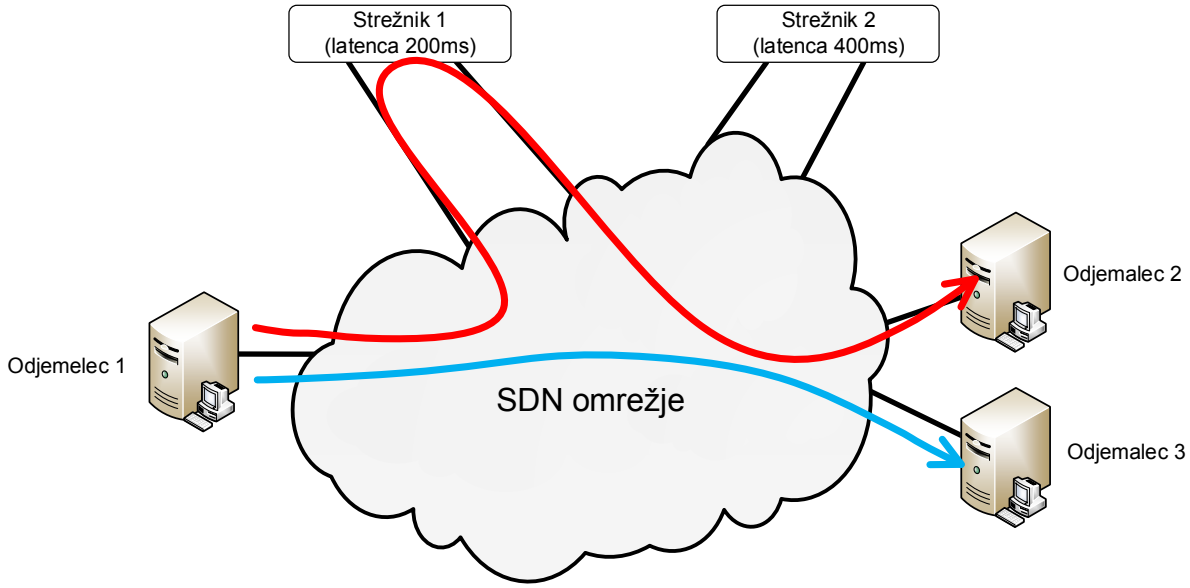
```
$ nova boot --flavor cirros.nano --image $(nova image-list | grep 'cirros'\
| awk '{print $2}' | tail -1) --nic net-id=$(neutron net-list | grep –w \
vx-net | awk '{print $2}') Streznik1 --availability_zone=nova:openstack03

$ nova boot --flavor cirros.nano --image $(nova image-list | grep 'cirros'\
| awk '{print $2}' | tail -1) --nic net-id=$(neutron net-list | grep –w \
vx-net | awk '{print $2}') Streznik2 --availability_zone=nova:openstack04
```

Če ni bilo storjenih napak, se nam na konzoli, potem ko smo zagnali ukaze v njenem kontrolnem vozlišču na oblačni platformi, prikaže tabela kreiranih instanc usmerjevalnika in navideznega omrežja.

```
$[root@openstack01 ~(keystone_admin)]#nova list
+------------+--------+------------+-------------+--------------------+
| Name       | Status | Task State | Power State | Networks           |
+------------+--------+------------+-------------+--------------------+
| Odjemalec1 | ACTIVE | -          | Running     | vx-net=10.0.0.2    |
| Odjemalec2 | ACTIVE | -          | Running     | vx-net=10.0.0.4    |
| Odjemalec3 | ACTIVE | -          | Running     | vx-net=10.0.0.5    |
| Streznik1  | ACTIVE | -          | Running     | vx-net=10.0.0.6    |
| Streznik2  | ACTIVE | -          | Running     | vx-net=10.0.0.7    |
+------------+--------+------------+-------------+--------------------+


$[root@openstack01 ~(keystone_admin)]#neutron router-list
+--------+-----------------------+
| name   | external_gateway_info |
+--------+-----------------------+
| vx-rtr | null                  |
+--------+-----------------------+

$[root@openstack01 ~(keystone_admin)]#neutron net-list
+--------+-------------+
| name   | subnets     |
+--------+-------------+
| vx-net | 10.0.0.0/24 |
+--------+-------------+
```

5.2.1 Prvi primer uporabe

V prvem primer bomo zgradili navidezno omrežje tokov OF in nato poslali pakete ICMP od prvega do drugega odjemalca preko prvega strežnika, ki ima funkcijo zakasnitve pošiljanja paketov (200 milisekund). Nato bomo poslali paket ICMP neposredno od prvega do tretjega odjemalca, ne da bi paket obšel prvi strežnik (pošiljanje brez zakasnitve).



Slika 5.6: Primer prometa skozi prvi strežnik

Za namen nastavitev VTN-okolja smo izdelali `bash` skripto `set_of_rules.sh`, ki za parameter tretira datoteko, v kateri so nanizani ukazi, ki prožijo zahteve HTTP nad vmesnikom REST API aplikacije VTN kontrolnika ODL. Skripte so prikazane v prilogi magistrskega dela. Za vzpostavitev poti med prvim in drugim odjemalcem preko prvega zakasnitvenega strežnika z zakasnitvijo 200 milisekund moramo na konzoli kontrolnega vozlišča oblačne platforme, kjer smo shranili spodaj uporabljene skripte, zagnati ukaze:

```
$./set_of_rules.sh kreiranje_VTN_infra.txt
$./set_of_rules.sh filter_odjemalec2.txt
$./set_of_rules.sh povezovanje_na_storitev1.txt
```

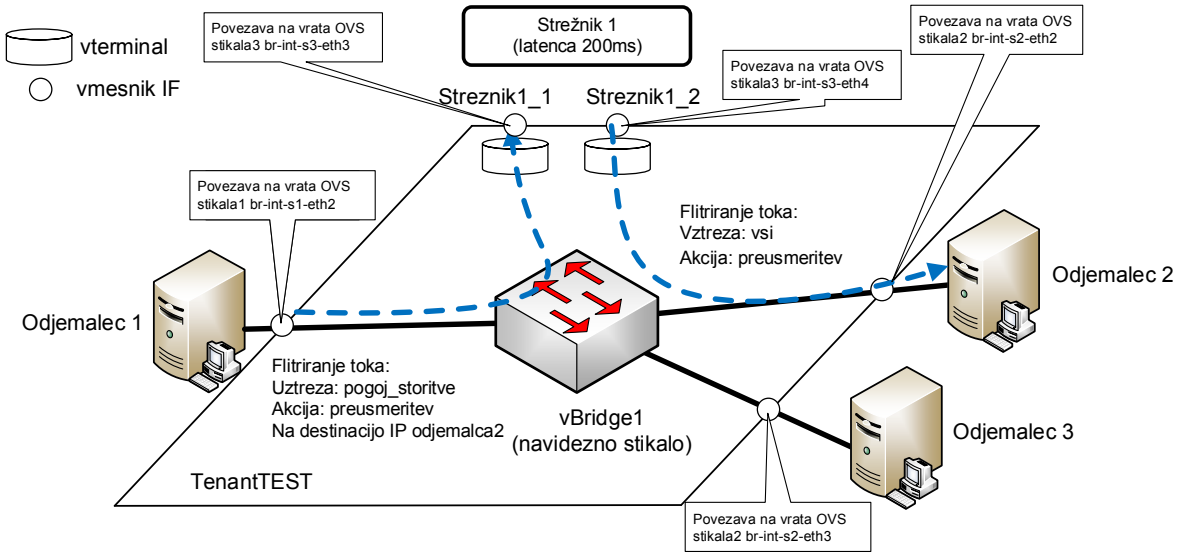
Zgornji ukazi kreirajo navidezno raven z imenom TennantTEST na ravni VTN v kontrolniku ODL. Nato kreirajo navidezni terminal vTerminala in vmesnike ter jih povežejo s stikalom OVS na omrežnih vozliščih oblačne platforme. Nato postavijo pravila, ki vse omrežne pakete preusmerijo na prvi strežnik, in nato posredujejo promet do drugega odjemalca. Za lažjo predstavo akcij, ki se izvedejo s skripto `set_of_rules.sh`, si lahko pogledamo sliko 5.7.

Za simulacijo zakasnitve 200 milisekund prvega strežnika pa moramo izvesti sledeči ukaz na omrežnem vmesniku br-int-s3-eth3 OVS-stikala na tretjem omrežnem vozlišču oblačne platforme OpenStack:

```
$ tc qdisc add dev br-int-s3-eth3 root netem delay 200ms
```

Podoben ukaz izvedemo za simulacijo zakasnitve 400 milisekund drugega strežnika na omrežnem vmesniku br-int-s4-eth4 OVS-stikala na četrtem omrežnem vozlišču oblačne platforme OpenStack:

```
$ tc qdisc add dev br-int-s4-eth4 root netem delay 400ms
```



Slika 5.7: Potek toka paketov ICMP skozi prvi strežnik

Testiramo novozgrajeno pot s pošiljanjem paketov ICMP od prvega do drugega odjemalca na konzoli prvega odjemalca (testiranje izvajamo z orodjema `ping` in `tracepath`, ki sta nameščena na operacijskem sistem Linux):

```
$ ping odjemalec2
PING 10.0.0.4 (10.0.0.4) 56(84) bytes of data.
64 bytes from 10.0.0.4: icmp_seq=1 ttl=64 time=209 ms
64 bytes from 10.0.0.4: icmp_seq=2 ttl=64 time=201 ms
64 bytes from 10.0.0.4: icmp_seq=3 ttl=64 time=200 ms
64 bytes from 10.0.0.4: icmp_seq=4 ttl=64 time=208 ms

$ tracepath odjemalec2
1?: [LOCALHOST]                pmtu 1500
1:  10.0.0.4                   201.024ms reached
1:  10.0.0.4                   200.717ms reached
```

Kot lahko opazimo, vsak paket ICMP za pot potrebuje približno 200 milisekund. Ko test pošiljanja ponovimo od prvega do tretjega odjemalca, lahko opazimo, da so časi potovanja paketa dosti manjši:

```
$ ping odjemalec3
PING 10.0.0.5 (10.0.0.5) 56(84) bytes of data.
64 bytes from 10.0.0.5: icmp_seq=1 ttl=64 time=0.030 ms
64 bytes from 10.0.0.5: icmp_seq=2 ttl=64 time=0.035 ms
64 bytes from 10.0.0.5: icmp_seq=3 ttl=64 time=0.042 ms
64 bytes from 10.0.0.5: icmp_seq=4 ttl=64 time=0.041 ms
```

Do razlik v potovalnem času prihaja, ker omrežni paketi, ki so namenjeni proti drugemu odjemalcu, ustrezajo pogoju `pogoj_storitve`, ki preusmeri pakete v prvi zakasnitveni strežnik. Med samim pošiljanjem vklopimo zajem paketov na konzoli tretjega omrežnega vozlišču oblačne platforme in čakamo na prispele ICMP-pakete na vmesniku `br-int-s3-eth4` stikala OVS:

```
$ tcpdump -i br-int-s3-eth4 -v icmp
---------------------- Rezultat zajema 4 ICMP paketov ----------------------
02:40:46.992146 IP (tos 0x0, ttl 64, id 47450, offset 0, flags [DF], proto
ICMP (1), length 84)
    10.0.0.2 > 10.0.0.4: ICMP echo request, id 13859, seq 193, length 64
02:40:46.992178 IP (tos 0x0, ttl 64, id 6665, offset 0, flags [none], proto
ICMP (1), length 84)
```

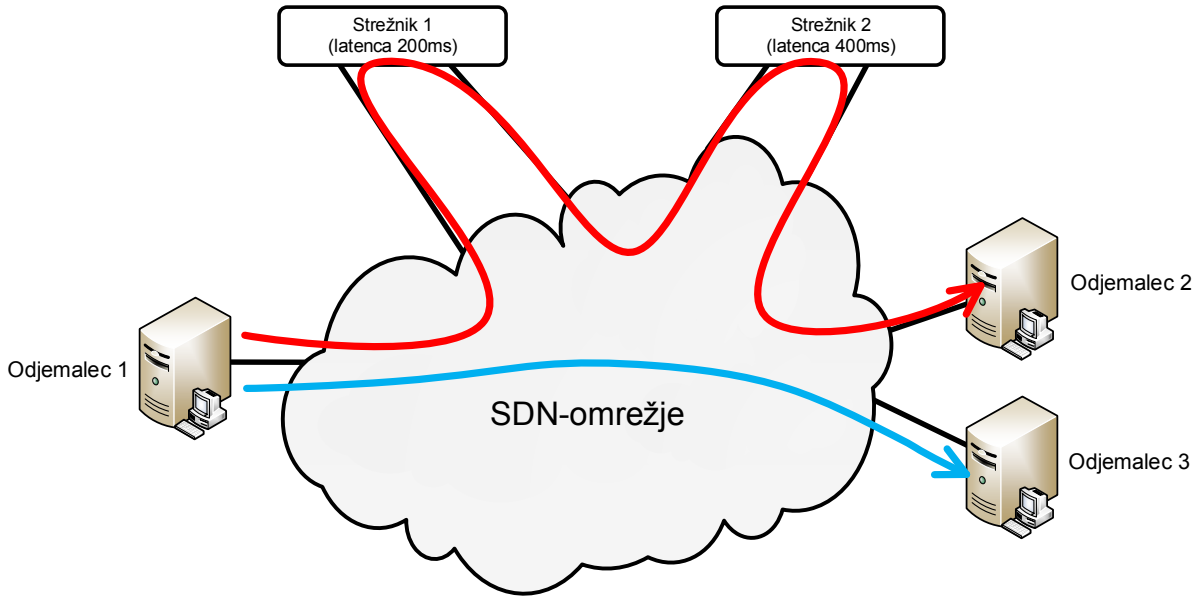
```
    10.0.0.4 > 10.0.0.2: ICMP echo reply, id 13859, seq 193, length 64
02:40:47.990898 IP (tos 0x0, ttl 64, id 47681, offset 0, flags [DF], proto
ICMP (1), length 84)
    10.0.0.2 > 10.0.0.4: ICMP echo request, id 13859, seq 194, length 64
02:40:47.990934 IP (tos 0x0, ttl 64, id 6758, offset 0, flags [none], proto
ICMP (1), length 84)
    10.0.0.4 > 10.0.0.2: ICMP echo reply, id 13859, seq 194, length 64
--------------------------------------------------------------------------

$ tshark -i br-int-s3-eth4 -O icmp
--------------------- Rezultat zajema dveh ICMP paketov -------------------
Frame 1: 98 bytes on wire (784 bits), 98 bytes captured (784 bits) on interface
0
Ethernet II, Src: e6:cb:46:32:0d:ae (e6:cb:46:32:0d:ae), Dst:
06:b0:41:c9:c1:5d (06:b0:41:c9:c1:5d)
Internet Protocol Version 4, Src: 10.0.0.2 (10.0.0.2), Dst: 10.0.0.4
(10.0.0.4)
Internet Control Message Protocol
    Type: 8 (Echo (ping) request)
    Code: 0
    Checksum: 0xa71e [correct]
    Identifier (BE): 13859 (0x3623)
    Identifier (LE): 9014 (0x2336)
    Sequence number (BE): 240 (0x00f0)
    Sequence number (LE): 61440 (0xf000)
    Timestamp from icmp data: Jun 14, 2015 02:41:34.000000000 PDT
    [Timestamp from icmp data (relative): 0.784654000 seconds]
    Data (48 bytes)

0000  e4 f8 0b 00 00 00 00 00 10 11 12 13 14 15 16 17   ................
0010  18 19 1a 1b 1c 1d 1e 1f 20 21 22 23 24 25 26 27   ........ !"#$%&'
0020  28 29 2a 2b 2c 2d 2e 2f 30 31 32 33 34 35 36 37   ()*+,-./01234567
        Data: e4f80b000000000010111213141516171819a1b1c1d1e1f...
        [Length: 48]
----------------------
Frame 2: 98 bytes on wire (784 bits), 98 bytes captured (784 bits) on interface
0
Ethernet II, Src: 06:b0:41:c9:c1:5d (06:b0:41:c9:c1:5d), Dst:
e6:cb:46:32:0d:ae (e6:cb:46:32:0d:ae)
Internet Protocol Version 4, Src: 10.0.0.4 (10.0.0.4), Dst: 10.0.0.2
(10.0.0.2)
Internet Control Message Protocol
    Type: 0 (Echo (ping) reply)
    Code: 0
    Checksum: 0xa71e [correct]
    Identifier (BE): 13859 (0x3623)
    Identifier (LE): 9014 (0x2336)
    Sequence number (BE): 240 (0x00f0)
    Sequence number (LE): 61440 (0xf000)
    [Request frame: 8]
    [Response time: 0.036 ms]
    Timestamp from icmp data: Jun 14, 2015 02:41:34.000000000 PDT
    [Timestamp from icmp data (relative): 0.784858000 seconds]
    Data (48 bytes)

0000  97 59 00 00 00 00 00 00 10 11 12 13 14 15 16 17   .Y..............
0010  18 19 1a 1b 1c 1d 1e 1f 20 21 22 23 24 25 26 27   ........ !"#$%&'
0020  28 29 2a 2b 2c 2d 2e 2f 30 31 32 33 34 35 36 37   ()*+,-./01234567
        Data: 97590000000000001011121314151617181 91a1b1c1d1e1f...
        [Length: 48]
--------------------------------------------------------------------------
```

Za sam zajem paketov po protokolu ICMP smo uporabili odprtokodni programski orodji `tcpdump` in `tshark`, ki tečeta v okolju Linux in sta namenjeni analizi omrežnega prometa. Z rezultati, ki smo jih zajeli z orodjem `tcpdump`, lahko vidimo, da prvi odjemalec (10.0.0.2) pošilja paket ICMP drugemu odjemalcu (10.0.0.2) in nato prejme odgovor drugega odjemalca. Za dodatne informacije smo zajeli pakete še z orodjem `tshark`, ki pokaže glavo posameznih omrežnih ravni referenčnega modela OSI. Kot lahko vidimo, sta bila zajeta dva paketa ICMP. V glavah paketov ICMP lahko vidimo, da gre za isto omrežno sejo zahteve (request) in odgovora (reply) med prvim in drugim odjemalcem.

S tem primerom uporabe smo pokazali, kako enostavno se vzpostavi navidezno omrežno stikalo in povezave med strežniki v SDN-omrežjih. S takim načinom lahko administrator omrežja v oblaku zelo hitro postavi nova omrežja, ki ustrezajo uporabniškim potrebam.

5.2.2 Drugi primer uporabe

V drugem primeru uporabe bomo vključili še drugi strežnik, ki ima zakasnitev 400 milisekund. Ko bomo s tokovnimi pravili spremenili pot paketov, tako da bodo obšli še drugi strežnik, se bo čas potovanja paketa od izvora do ponora povečal na 600 milisekund (200 ms + 400 ms). Za lažjo predstavo je na voljo slika 5.8.

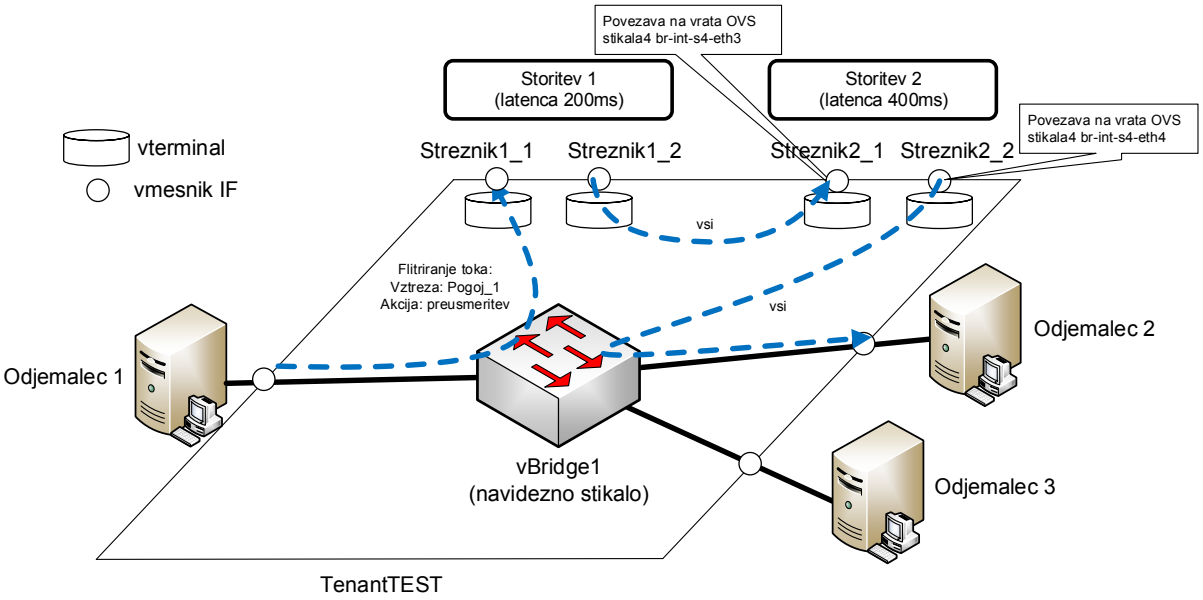


Slika 5.8: Primer prometa skozi prvi in drugi strežnik

Nato zaženemo ukaz na kontrolnem vozlišču oblačne platforme, ki bo v obstoječi že zgrajen tok dodal drugo storitev z zakasnitvijo 400 milisekund:

```
$./set_of_rules.sh povezovanje_na_storitev2.txt
```

Za lažjo predstavo akcij, ki jih izvede skripta `set_of_rules.sh`, si lahko ogledamo sliko 5.9. Kot lahko vidimo, se s tokovnimi pravili kreira nova povezava preko drugega strežnika. Določi se nov pogoj `Pogoj_1`, ki preusmerja pakete, ki imajo za destinacijo drugega odjemalca po novozgrajeni poti.



Slika 5.9: Potek toka paketov ICMP skozi obe storitvi

Ponovno testiramo dodano novo pot s pošiljanjem paketov ICMP do drugega odjemalca na konzoli prvega odjemalca:

```
$ ping odjemalec2
PING 10.0.0.4 (10.0.0.4) 56(84) bytes of data.
64 bytes from 10.0.0.4: icmp_seq=35 ttl=64 time=607 ms
64 bytes from 10.0.0.4: icmp_seq=36 ttl=64 time=604 ms
64 bytes from 10.0.0.4: icmp_seq=37 ttl=64 time=601 ms
64 bytes from 10.0.0.4: icmp_seq=38 ttl=64 time=613 ms
```

```
$ tracepath odjemalec2
1?: [LOCALHOST]                      pmtu 1500
1:  10.0.0.4                        601.490ms reached
1:  10.0.0.4                        601.580ms reached
```

Kot vidimo, se poveča potovalna pot ICMP paketa na približno 600 milisekund. Za razliko od tega pošiljanje paketov ICMP do tretjega odjemalca traja precej manj časa, saj ne obide prvega in drugega strežnika (glej sliko 5.8 in 5.9):

```
$ ping odjemalec3
PING 10.0.0.5 (10.0.0.5) 56(84) bytes of data.
64 bytes from 10.0.0.5: icmp_seq=1 ttl=64 time=0.031 ms
64 bytes from 10.0.0.5: icmp_seq=2 ttl=64 time=0.211 ms
64 bytes from 10.0.0.5: icmp_seq=3 ttl=64 time=0.047 ms
64 bytes from 10.0.0.5: icmp_seq=4 ttl=64 time=0.052 ms
```

Med samim pošiljanjem ponovno zajemamo pakete na konzoli četrtega omrežnega vozlišča oblačne platforme in čakamo na prispele ICMP-pakete na vmesniku `br-int-s4-eth4` stikala OVS:

```
$ tcpdump -i br-int-s4-eth3 -v icmp

---------------------- Rezultat zajema 4 ICMP paketov ----------------------
02:39:53.956248 IP (tos 0x0, ttl 64, id 41399, offset 0, flags [DF], proto
ICMP (1), length 84)
    10.0.0.2 > 10.0.0.4: ICMP echo request, id 13859, seq 140, length 64
02:39:53.956287 IP (tos 0x0, ttl 64, id 173, offset 0, flags [none], proto
ICMP (1), length 84)
    10.0.0.4 > 10.0.0.2: ICMP echo reply, id 13859, seq 140, length 64
02:39:54.955224 IP (tos 0x0, ttl 64, id 41643, offset 0, flags [DF], proto
ICMP (1), length 84)
    10.0.0.2 > 10.0.0.4: ICMP echo request, id 13859, seq 141, length 64
02:39:54.955251 IP (tos 0x0, ttl 64, id 420, offset 0, flags [none], proto
ICMP (1), length 84)
    10.0.0.4 > 10.0.0.2: ICMP echo reply, id 13859, seq 141, length 64
------------------------------------------------------------------------

$ tshark -i br-int-s4-eth4 -O icmp

---------------------- Rezultat zajema 2 ICMP paketov ----------------------
Frame 1: 98 bytes on wire (784 bits), 98 bytes captured (784 bits) on
interface 0
Ethernet II, Src: e6:cb:46:32:0d:ae (e6:cb:46:32:0d:ae), Dst:
06:b0:41:c9:c1:5d (06:b0:41:c9:c1:5d)
Internet Protocol Version 4, Src: 10.0.0.2 (10.0.0.2), Dst: 10.0.0.4
(10.0.0.4)
Internet Control Message Protocol
    Type: 8 (Echo (ping) request)
    Code: 0
    Checksum: 0x9565 [correct]
    Identifier (BE): 28130 (0x6de2)
    Identifier (LE): 57965 (0xe26d)
    Sequence number (BE): 69 (0x0045)
    Sequence number (LE): 17664 (0x4500)
    Timestamp from icmp data: Jun 14, 2015 01:17:48.000000000 PDT
    [Timestamp from icmp data (relative): 0.070023000 seconds]
    Data (48 bytes)

0000  e1 7a 05 00 00 00 00 00 10 11 12 13 14 15 16 17   .z..............
0010  18 19 1a 1b 1c 1d 1e 1f 20 21 22 23 24 25 26 27   ........ !"#$%&'
0020  28 29 2a 2b 2c 2d 2e 2f 30 31 32 33 34 35 36 37   ()*+,-./01234567
        Data: e17a0500000000001011121314151617181 91a1b1c1d1e1f...
        [Length: 48]

------------------------------------------------
Frame 11: 98 bytes on wire (784 bits), 98 bytes captured (784 bits) on
interface 0
Ethernet II, Src: 06:b0:41:c9:c1:5d (06:b0:41:c9:c1:5d), Dst:
e6:cb:46:32:0d:ae (e6:cb:46:32:0d:ae)
Internet Protocol Version 4, Src: 10.0.0.4 (10.0.0.4), Dst: 10.0.0.2
(10.0.0.2)
Internet Control Message Protocol
    Type: 0 (Echo (ping) reply)
    Code: 0
    Checksum: 0x9565 [correct]
    Identifier (BE): 28130 (0x6de2)
    Identifier (LE): 57965 (0xe26d)
    Sequence number (BE): 69 (0x0045)
    Sequence number (LE): 17664 (0x4500)
    [Request frame: 10]
    [Response time: 0.033 ms]
```

```
    Timestamp from icmp data: Jun 14, 2015 01:17:48.000000000 PDT
    [Timestamp from icmp data (relative): 0.070057000 seconds]
    Data (48 bytes)

0000  55 11 01 00 00 00 00 00 10 11 12 13 14 15 16 17   U...............
0010  18 19 1a 1b 1c 1d 1e 1f 20 21 22 23 24 25 26 27   ........ !"#$%&'
0020  28 29 2a 2b 2c 2d 2e 2f 30 31 32 33 34 35 36 37   ()*+,-./01234567
       Data: 55110100000000001011121314151617181 91a1b1c1d1e1f...
       [Length: 48]
------------------------------------------------------------------------
```

Zajem podatkov s programskim orodjem `tcpdump` in `tshark` poda podobne rezultate kot orodji pri prvem primeru uporabe, zato dodatna razlaga tu ni potrebna.

Praktična uporaba lastnosti takega usmerjanja pride v poštev v primerih, ko želim opravljati vzdrževalna dela na določenem segmentu omrežja, hkrati pa moramo zagotavljati neprekinjeno delovanje strežniških storitev. Druga uporabnost take lastnosti je lahko pri selitvi posameznih skupin uporabnikov v določeni organizaciji na drugo oddaljeno lokacijo.

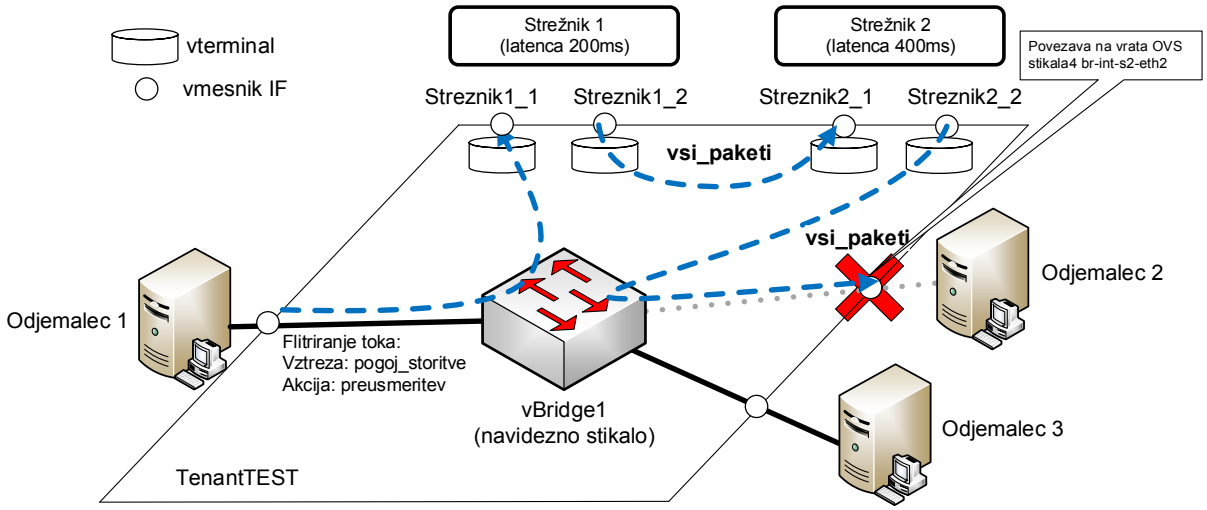
5.2.3 Tretji primer uporabe

V tretjem primeru bomo pokazali, kako lahko onemogočimo vmesnik in s tem izoliramo odjemalca. Take primere v praksi pogosto srečujemo na fizičnih omrežjih, kjer pride do omrežne zanke zaradi napačno nastavljenih povezanih stikal. Seveda pa grožnje ne obstajajo samo na omrežnem nivoju, ampak tudi na aplikativnem. Eden primerov tega v praksi je na primer z virusom okužen računalnik, ki lahko povzroči veliko škodo, zato ga želimo začasno izločiti iz omrežja.

Za odstranitev omrežne zanke med stikali že dolgo obstaja protokol, ki to odpravi (STP – Spanning Tree Protocol), medtem ko reševanje problema z virusom okuženega računalnika v klasičnem omrežju ni tako enostavno. S sistemom za detekcijo napadov (sistem IDS) lahko odkrijemo računalnike, ki so okuženi z virusom, in informacijo o tem posredujemo vzdrževalcu omrežja, ki bo preko ODL-kontrolnika izoliral okužene računalnike v omrežju. V našem primeru to pomeni, da izklopimo vmesnik na navideznem stikalu (vBridge1), ki smo ga prej kreirali v okolju VTN (glej sliko 5.10).

Za opisan primer smo pripravili skripto za vklop in izklop vmesnika na navideznem stikalu vBridge1, na katerega je povezan drugi odjemalec. Za izklop vmesnika do drugega odjemalca zaženemo ukaz na konzoli kontrolnega vozlišča oblačne platforme:

```
$./set_of_rules.sh izklop_vmesnika_do_odjemalca2.txt
```



Slika 5.10: Odstranitev povezave do drugega odjemalca

Nato preverimo pot do drugega odjemalca s pošiljanjem paketa ICMP na konzoli prvega odjemalca:

```
$ ping odjemalec2
PING 10.0.0.4 (10.0.0.4) 56(84) bytes of data.
From 10.0.0.2 icmp_seq=1 Destination Host Unreachable
From 10.0.0.2 icmp_seq=2 Destination Host Unreachable
From 10.0.0.2 icmp_seq=3 Destination Host Unreachable
From 10.0.0.2 icmp_seq=4 Destination Host Unreachable
292 packets transmitted, 0 received, 100% packet loss, time 293189ms
```

Med samim pošiljanjem ponovno vklopimo zajem paketov na konzoli četrtega OVS-stikala, ki se nahaja na četrtem omrežnem vozlišču oblačne platforme (glej sliko 5.4), in čakamo na prispele pakete ICMP na omrežnem vmesniku br-int-s4-eth4:

```
$ tcpdump -i br-int-s4-eth4 -v icmp
listening on br-int-s4-eth4, link-type EN10MB (Ethernet), capture size
65535 bytes
0 packets captured
0 packets received by filter
0 packets dropped by kernel
```

Opazimo, da nismo zajeli nobenega paketa (kar kaže na to, da smo računalnik uspešno izključili iz omrežja), zato ponovimo postopek vzdolž toka povezave med prvim in drugim odjemalcem (glej sliko 5.10). Tako zajamemo pakete na konzoli tretjega OVS-stikala, ki se nahaja na tretjem omrežnem vozlišču oblačne platforme (glej sliko 5.4), in čakamo na prispele pakete ICMP na omrežnem vmesniku br-int-s3-eth4:

```
$ tcpdump -i br-int-s3-eth4 -v icmp
listening on br-int-s3-eth4, link-type EN10MB (Ethernet), capture size
65535 bytes
0 packets captured
0 packets received by filter
0 packets dropped by kernel
```

Ponovno ugotovimo, da nismo zajeli nobenega paketa, zato se ponovno premaknemo vzdolž toka povezave med prvim in drugim odjemalcem. Tako zajamemo pakete na konzoli pravega OVS-stikala,

ki se nahaja na prvem omrežnem vozlišču oblačne platforme (glej sliko 5.4), in čakamo na prispele pakete ICMP na omrežnem vmesniku br-int-s1-eth2:

```
$ tcpdump –i br-int-s1-eth2 –v icmp
listening on br-int-s1-eth2, link-type EN10MB (Ethernet), capture size
65535 bytes
0 packets captured
0 packets received by filter
0 packets dropped by kernel
```

Ugotovimo, da ni bil poslan noben ICMP-paket. Zato pošiljanje ponovimo. Tokrat oddajamo pakete ARP na nižjem nivoju omrežnega modela OSI:

```
$ arping -I eth0 odjemalec2
ARPING 10.0.0.4 from 10.0.0.2 eth0
Sent 53 probes (53 broadcast(s))
Received 0 response(s)
```

Ponovno zajamemo pakete na začetku povezave med prvim in drugim odjemalcem na konzoli prvega OVS-stikala, ki se nahaja na prvem omrežnem vozlišču oblačne platforme (glej sliko 5.4), in oddaja ARP-pakete na omrežnem vmesniku br-int-s1-eth2:

```
$ tcpdump -i br-int-s1-eth2 -v arp

tcpdump: listening on s1-eth2, link-type EN10MB (Ethernet), capture size
65535 bytes
04:42:24.666254 ARP, Ethernet (len 6), IPv4 (len 4), Request who-has
10.0.0.4 tell 10.0.0.2, length 28
04:42:25.665971 ARP, Ethernet (len 6), IPv4 (len 4), Request who-has
10.0.0.4 tell 10.0.0.2, length 28
04:42:26.666232 ARP, Ethernet (len 6), IPv4 (len 4), Request who-has
10.0.0.4 tell 10.0.0.2, length 28
04:42:27.666026 ARP, Ethernet (len 6), IPv4 (len 4), Request who-has
10.0.0.4 tell 10.0.0.2, length 28


$ tshark -i br-int-s1-eth2 -O icmp

-------- Rezultat zajema 4 ARP paketov na vmesniku br-int-s1-eth2 ---------
1 Frame 68: 42 bytes on wire (336 bits), 42 bytes captured (336 bits) on
interface 0
Ethernet II, Src: e6:cb:46:32:0d:ae (e6:cb:46:32:0d:ae), Dst:
06:b0:41:c9:c1:5d (06:b0:41:c9:c1:5d)
Address Resolution Protocol (request)

2 Frame 69: 42 bytes on wire (336 bits), 42 bytes captured (336 bits) on
interface 0
Ethernet II, Src: e6:cb:46:32:0d:ae (e6:cb:46:32:0d:ae), Dst:
06:b0:41:c9:c1:5d (06:b0:41:c9:c1:5d)
Address Resolution Protocol (request)

3 Frame 69: 42 bytes on wire (336 bits), 42 bytes captured (336 bits) on
interface 0
Ethernet II, Src: e6:cb:46:32:0d:ae (e6:cb:46:32:0d:ae), Dst:
06:b0:41:c9:c1:5d (06:b0:41:c9:c1:5d)
Address Resolution Protocol (request)
```

```
4 Frame 69: 42 bytes on wire (336 bits), 42 bytes captured (336 bits) on
interface 0
Ethernet II, Src: e6:cb:46:32:0d:ae (e6:cb:46:32:0d:ae), Dst:
06:b0:41:c9:c1:5d (06:b0:41:c9:c1:5d)
Address Resolution Protocol (request)
-------------------------------------------------------------------------
```

Ugotovimo, da drugi odjemalec pošlje ARP-zahtevo (paket) skozi vmesnik `br-int-s1-eth2`, vendar nanje ne dobi nobenega odgovora. Zaradi izklopa vmesnika, na katerega je priklopljen drugi odjemalec na navideznem stikalu vBridge1, pošiljanje paketov ICMP ni več mogoče. Ker v ozadju teče protokol LLDP, ki je odgovoren za osveževanje topologije omrežja (glej sliko 5.2) v navideznem stikalu, se spremeni tudi vsebina MAC tabele (glej sliko 5.3), tako da se izbriše MAC naslov drugega odjemalca.

```
$ tshark -i br-int-s1-eth2 -O icmp

-------- Rezultat zajema 2 LLDP paketov na vmesniku br-int-s1-eth2 --------
2 Frame 8: 85 bytes on wire (680 bits), 85 bytes captured (680 bits) on
interface 0
Ethernet II, Src: e6:cb:46:32:0d:ae (e6:cb:46:32:0d:ae), Dst:
06:b0:41:c9:c1:5d (06:b0:41:c9:c1:5d)
Link Layer Discovery Protocol

5 Frame 10: 85 bytes on wire (680 bits), 85 bytes captured (680 bits) on
interface 0
Ethernet II, Src: e6:cb:46:32:0d:ae (e6:cb:46:32:0d:ae), Dst:
06:b0:41:c9:c1:5d (06:b0:41:c9:c1:5d)
Link Layer Discovery Protocol
-------------------------------------------------------------------------
```

S tem si lahko razložimo, zakaj pošiljanje paketov ICMP v prvem poskusu ni bilo uspešno. Razlog je, da se je izbrisala vsa sled za drugim odjemalcem v MAC tabeli v navideznem stikalu vBridge1, kar je bil tudi naš namen. Medtem lahko na primer administrator uspešno očisti računalnik prejšnje okužbe in ga spet vključi v omrežje. Če ponovno vklopimo vmesnik na navideznem stikalu vBridge1, na katerega je priključen drugi odjemalec, se MAC tabela posodobi. Za vklop vmesnika do drugega odjemalca zaženemo ukaz na konzoli kontrolnega vozlišča oblačne platforme:

```
$ ./set_of_rules.sh  vklop_vmesnika_do_odjemalca2.txt
```

S tem spet vzpostavimo delujočo povezavo od prvega do drugega odjemalca:

```
$ ping odjemalec2
PING 10.0.0.4 (10.0.0.4) 56(84) bytes of data.
64 bytes from 10.0.0.4: icmp_seq=35 ttl=64 time=607 ms
64 bytes from 10.0.0.4: icmp_seq=36 ttl=64 time=604 ms
64 bytes from 10.0.0.4: icmp_seq=37 ttl=64 time=601 ms
64 bytes from 10.0.0.4: icmp_seq=38 ttl=64 time=613 ms
```

Naredimo še poskus ponovnega zajema paketov na konzoli četrtega OVS-stikala, ki se nahaja na četrtem omrežnem vozlišču oblačne platforme (glej sliko 5.4), in čakamo na prispele ICMP-pakete na omrežnem vmesniku br-int-s4-eth4:

```
$ tshark -i br-int-s4-eth4 -O icmp
```

```
--------------------- Rezultat zajema 2 ICMP paketov ----------------------
Frame 1: 98 bytes on wire (784 bits), 98 bytes captured (784 bits) on
interface 0
Ethernet II, Src: e6:cb:46:32:0d:ae (e6:cb:46:32:0d:ae), Dst:
06:b0:41:c9:c1:5d (06:b0:41:c9:c1:5d)
Internet Protocol Version 4, Src: 10.0.0.2 (10.0.0.2), Dst: 10.0.0.4
(10.0.0.4)
Internet Control Message Protocol
    Type: 8 (Echo (ping) request)
    Code: 0
    Checksum: 0x8fde [correct]
    Identifier (BE): 24157 (0x3623)
    Identifier (LE): 8012 (0x2336)
    Sequence number (BE): 344 (0x0158)
    Sequence number (LE): 22439 (0x5801)
    Timestamp from icmp data: Jun 14, 2015 04:13:37.000000000 PDT
    [Timestamp from icmp data (relative): 0.070023000 seconds]
    Data (48 bytes)

0000  e1 7a 05 00 00 00 00 00 10 11 12 13 14 15 16 17   .z..............
0010  18 19 1a 1b 1c 1d 1e 1f 20 21 22 23 24 25 26 27   ........ !"#$%&'
0020  28 29 2a 2b 2c 2d 2e 2f 30 31 32 33 34 35 36 37   ()*+,-./01234567
       Data: e17a0500000000001011121314151617181 91a1b1c1d1e1f...
       [Length: 48]

------------------------------------------------
Frame 11: 98 bytes on wire (784 bits), 98 bytes captured (784 bits) on
interface 0
Ethernet II, Src: 06:b0:41:c9:c1:5d (06:b0:41:c9:c1:5d), Dst:
e6:cb:46:32:0d:ae (e6:cb:46:32:0d:ae)
Internet Protocol Version 4, Src: 10.0.0.4 (10.0.0.4), Dst: 10.0.0.2
(10.0.0.2)
Internet Control Message Protocol
    Type: 0 (Echo (ping) reply)
    Code: 0
    Checksum: 0x8fde [correct]
    Identifier (BE): 24157 (0x3623)
    Identifier (LE): 8012 (0x2336)
    Sequence number (BE): 344 (0x0158)
    Sequence number (LE): 22439 (0x5801)
    [Request frame: 10]
    [Response time: 0.033 ms]
    Timestamp from icmp data: Jun 14, 2015 04:13:37.000000000 PDT
    [Timestamp from icmp data (relative): 0.070057000 seconds]
    Data (48 bytes)

0000  55 11 01 00 00 00 00 00 10 11 12 13 14 15 16 17   U...............
0010  18 19 1a 1b 1c 1d 1e 1f 20 21 22 23 24 25 26 27   ........ !"#$%&'
0020  28 29 2a 2b 2c 2d 2e 2f 30 31 32 33 34 35 36 37   ()*+,-./01234567
       Data: 551101000000000010111213141516171 8191a1b1c1d1e1f...
       [Length: 48]
-------------------------------------------------------------------------
```

Kot lahko vidimo iz rezultatov, pridobljenih z orodjema `tcpdump` in `tshark`, se je povezava med drugim odjemalcem in navideznim stikalom (vBridge1) ponovno vzpostavila in dinamično spremenila MAC tabelo, tako da ponovno vsebuje MAC naslov drugega odjemalca. Drugi odjemalec je s tem spet vključen v omrežje.

Z opisanimi primeri smo pokazali, kako lahko relativno z malo truda preprosto vplivamo na promet v omrežju in ga po potrebi preusmerjamo, ali pa določene naprave odklapljamo in spet priklapljamo v omrežje. Z uporabo teh lastnosti pa lahko pripomoremo k boljšemu upravljanju omrežja v oblaku. Tako lahko ponudnik oblačnih storitev svojim potrošnikom (uporabnikom) dinamično po potrebi kreira, prilagaja in odstranjuje topologijo najetega omrežja v oblaku.

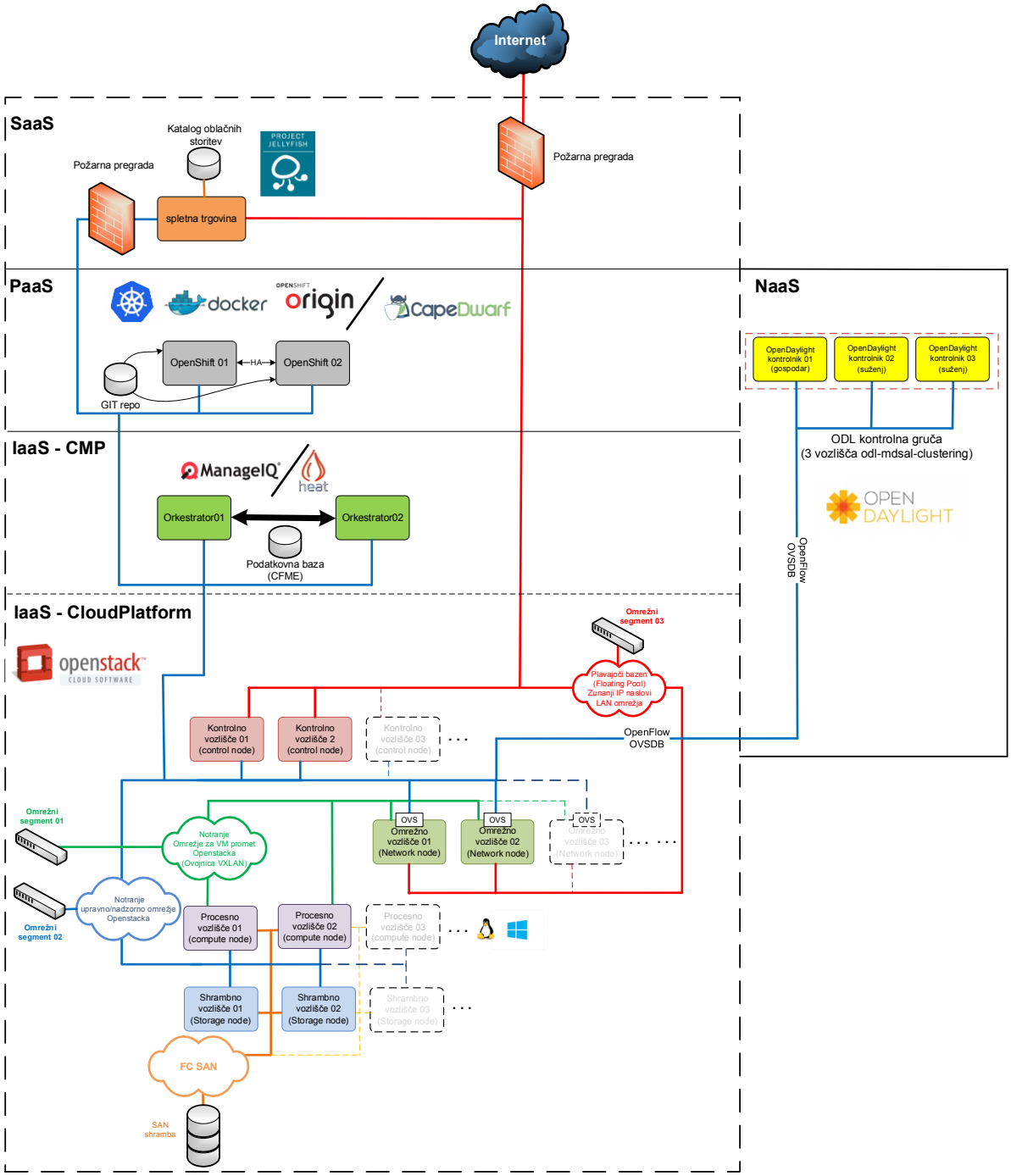
6 Zaključek

V računalništvu so uporabniške storitve bistvenega pomena. Vsak, ki uporablja računalnik, to počne, ker želi dostopati do neke storitve, ali pa želi svoje storitve ponuditi drugim uporabnikom. Dobro vemo tudi, da na uspešnost trženja storitev vplivajo predvsem uporabnost, cena in dostopnost storitve. S prihodom oblačne tehnologije sta se dostopnost in cenovna dosegljivost računalniških storitev močno izboljšali. S pomočjo oblaka lahko svoje storitve ponujamo hitreje in enostavneje. Oblačna arhitektura mora biti zasnovana tako, da se lahko dinamično prilagaja storitvam, ki tečejo na njej, kar pomeni, da se mora oblak prilagajati uporabniškim storitvam. Seveda pa za hitro in enostavno ponudbo računalniških storitev v oblaku stoji ogromna oblačna infrastruktura, ki ima zelo kompleksno arhitekturo.

Ker je oblačna arhitektura zasnovana tako, da se dinamično prilagaja potrebam storitev, je koncept tradicionalnega klasičnega omrežja postal neprimeren. S tem se pokaže potreba po pametnih računalniških omrežjih, ki se znajo prilagoditi potrebam posameznih storitev v oblaku. Programsko določena omrežja lahko ustrežejo tem potrebam in s pomočjo tehnologije virtualizacije omrežja zagotavljajo dinamično spreminjanje topologije in optimizacijo omrežnih poti za boljšo dostopnost storitev. Da pa lahko izdelamo takšno arhitekturo, je potrebno izbrati pravilno oblačno platformo, ki se bo sposobna najbolje integrirati s programsko določenim omrežjem.

Namen magistrskega dela je bilo izdelati optimalno arhitekturo programsko določenega omrežja v oblaku in prikazati primer uporabnosti take arhitekture. V magistrskem delu smo za namene oblačne platforme preučili več odprtokodnih oblačnih platform. Ker se v postopku integracije oblačne platforme s programsko določenim omrežjem v največji meri dotikamo oblačnega modela arhitekture IaaS, smo kot najbolj primerno izbrali oblačno platformo OpenStack. Platforma OpenStack je najbolj prilagodljiva in podprta s strani razvijalcev različnih oblačnih rešitev ter se trenutno najbolj razvija. Programsko določena omrežja ločujejo kontrolni in podatkovni del omrežja ter s tem omogočajo boljše in enostavnejše upravljanje celotnega omrežja. Za kontrolni del omrežja smo za ta namen preučili nekaj odprtokodnih kontrolnikov. Najbolj primeren izmed njih je bil kontrolnik ODL, ki se integrira v oblačno platformo OpenStack in zna upravljati tako s fizičnimi kot tudi z navideznimi omrežnimi stikali. Kontrolnik je precej razširjen pri različnih globalnih proizvajalcih omrežne opreme. Kot smo prikazali v magistrskem delu, kontrolnik ponuja vrsto funkcionalnosti in možnost prilagoditve potrebam, ki jih zahtevajo omrežne oblačne storitve. Vsebuje pa tudi nekaj že pripravljenih aplikacij, ki omogočajo dinamične spremembe v topologiji omrežja v oblaku. Kot enega takšnih primerov smo predstavili aplikacijo VTN, s katero smo v magistrskem delu pokazali primer uporabe programsko določenih omrežnih storitev v oblaku. Za podatkovni del programsko določenega omrežja smo izbrali odprto stikalo OVS, s katerim smo upravljali preko protokola OF na kontrolniku ODL.

Oblačna arhitektura se trenutno najhitreje razvija in z uvedbo tehnologije kontejnerjev, kot jo ponuja koncept Docker, še bolj pridobiva na uporabnosti. S to tehnologijo oblačno storitev poponoma ločimo od infrastrukturnega nivoja in jo približamo aplikativnem. Tako dodamo dodatni nivo abstrakcije, ki omogoča večjo prenosljivost med različnimi oblačnimi platformami. Na podlagi tega predvidevamo, da se bo kontrolni del programsko določenega omrežja selil na višji nivo oblačne arhitekture (v oblačni model arhitekture PaaS), podatkovni del pa bo ostal na nivoju oblačne arhitekture IaaS. Na sliki 6.1 je prikazana idejna zasnova arhitekture z uporabo tehnologijo kontejnerjev in prikazuje vse arhitekturne modele oblaka (IaaS, PaaS, SaaS). S sliko 6.1 skušamo prikazati še preostala dva manjkajoča kosa, ki zaključujeta celotno piramido oblačne arhitekture.



Slika 6.1: Idejna zasnova arhitekture SDN-omrežja skozi vse modele arhitekture v oblaku

Na sliki 6.1 smo izbrali nekaj tehnologij na različnih nivojih, ki so trenutno najbolj uporabne za sestavo odprtega oblaka. Tako smo na nivoju IaaS dodali komponento orkestrator, ki olajša orkestracijo celotnega oblaka (uporaba programske opreme ManageIQ in OpenStack Heat). Na nivoju PaaS smo izbrali odprtokodno programsko opremo OpenShift Origin, ki temelji na tehnologiji kontejnerjev. Na najvišjem nivoju SaaS pa smo uporabili odprtokodno programsko opremo JellyFish, ki zagotavlja spletni portal in katalog storitev, preko katerega lahko uporabnik naroča oblačne storitve.

Programsko določeno omrežje v oblaku predstavlja dodatno raven kompleksnosti, in zaradi tega ni primerno za postavitev v majhnih okoljih, saj gre pri postavitvi takih omrežij za velike denarne in

časovne investicije. SDN najbolj ustrezajo ponudnikom telekomunikacijskih storitev v tujini, kot so ameriški AT&T, kitajski Unicom ali ruski MegaFon. Tehnologija SDN uporablja odprte sisteme in standardizirane komponente, kar lahko predstavlja izziv za proizvajalce. Ti morajo poskrbeti za podporo skupni uporabnosti omrežnih naprav, po drugi strani pa morajo stopiti v korak s časom in podpreti nove tehnologije. Proizvajalci omrežne in programske opreme ne bodo še tako kmalu sprejeli tehnologije SDN. Pred tem bo treba opraviti še veliko testiranj, so pa na trgu že nekateri, ki to počenejo. Eden takih je proizvajalec VMware s programsko platformo NSX.

SDN-omrežja lahko izboljšajo varnost v oblaku, vendar pa je njihova šibka točka prav SDN-kontrolnik. V primeru, da bi nekdo pridobil dostop do SDN-kontrolnika, bi lahko porušil celotno omrežje. Na podlagi tega lahko predvidevamo, da se bodo razvili novi napadi, ki bodo izkoriščali SDN-aplikacije za pridobitev dostopa do SDN-kontrolnika. Ob zavedanju tega bo treba na področju varnosti v programsko določenih omrežjih SDN narediti še veliko.

Za obvladovanje takega koncepta omrežja bo potrebno pridobiti tudi nekaj novega znanja, saj poznavanje omrežnega nivoja ne bo več zadostovalo. S tega vidika bodo morali omrežni strokovnjaki razširiti svoja znanja tudi na zgornji aplikativni ravni programsko določenega omrežja.

SDN-omrežja še niso tako pogosta v gospodarstvu, vendar pa vsak dan vzbujajo večjo pozornost. Razlog za to je v tem, da koncept klasičnih računalniških omrežji ne zadostuje več modelom oblačne arhitekture. Tako se kaže potreba po nadgradnji koncepta klasičnega računalniškega omrežje v oblaku z novim konceptom omrežij. Trenutno najbolj pravi koncept, ki ustreza tem potrebam, je koncept programsko določenih omrežij.

7 Priloga

7.1 Datoteka answers.cfg, ki jo zaženemo skupaj z orodjem packstack

```
################### Konfiguracijska datoteka answers.cfg ###################
[general]
CONFIG_SSH_KEY=/root/.ssh/id_rsa.pub
CONFIG_MARIADB_INSTALL=y
CONFIG_GLANCE_INSTALL=y
CONFIG_CINDER_INSTALL=y
CONFIG_NOVA_INSTALL=y
CONFIG_NEUTRON_INSTALL=y
CONFIG_HORIZON_INSTALL=y
CONFIG_SWIFT_INSTALL=y
CONFIG_CEILOMETER_INSTALL=y
CONFIG_HEAT_INSTALL=y
CONFIG_CLIENT_INSTALL=y
CONFIG_NTP_SERVERS=time.ijs.si
CONFIG_NAGIOS_INSTALL=n
EXCLUDE_SERVERS=
CONFIG_DEBUG_MODE=y
CONFIG_CONTROLLER_HOST=strežnik1
CONFIG_COMPUTE_HOSTS=strežnik1,strežnik2
CONFIG_NETWORK_HOSTS=strežnik1,strežnik2
CONFIG_VMWARE_BACKEND=n
CONFIG_UNSUPPORTED=n
CONFIG_VCENTER_HOST=
CONFIG_VCENTER_USER=
CONFIG_VCENTER_PASSWORD=
CONFIG_VCENTER_CLUSTER_NAME=
CONFIG_STORAGE_HOST=strežnik1
CONFIG_USE_EPEL=y
CONFIG_REPO=
CONFIG_RH_USER=
CONFIG_SATELLITE_URL=
CONFIG_RH_PW=
CONFIG_RH_OPTIONAL=y
CONFIG_RH_PROXY=
CONFIG_RH_PROXY_PORT=
CONFIG_RH_PROXY_USER=
CONFIG_RH_PROXY_PW=
CONFIG_SATELLITE_USER=
CONFIG_SATELLITE_PW=
CONFIG_SATELLITE_AKEY=
CONFIG_SATELLITE_CACERT=
CONFIG_SATELLITE_PROFILE=
CONFIG_SATELLITE_FLAGS=
CONFIG_SATELLITE_PROXY=
CONFIG_SATELLITE_PROXY_USER=
CONFIG_SATELLITE_PROXY_PW=
CONFIG_AMQP_BACKEND=rabbitmq
```

```
CONFIG_AMQP_HOST=strežnik1
CONFIG_AMQP_ENABLE_SSL=y
CONFIG_AMQP_ENABLE_AUTH=y
CONFIG_AMQP_NSS_CERTDB_PW=25a770f3cf82480bbc7663e098e14414
CONFIG_AMQP_SSL_PORT=5671
CONFIG_AMQP_SSL_CERT_FILE=/etc/pki/tls/certs/amqp_selfcert.pem
CONFIG_AMQP_SSL_KEY_FILE=/etc/pki/tls/private/amqp_selfkey.pem
CONFIG_AMQP_SSL_SELF_SIGNED=y
CONFIG_AMQP_AUTH_USER=amqp_user
CONFIG_AMQP_AUTH_PASSWORD=XXXXXXXX
CONFIG_MARIADB_HOST=strežnik1
CONFIG_MARIADB_USER=root
CONFIG_MARIADB_PW=XXXXXXXX
CONFIG_KEYSTONE_DB_PW=XXXXXXXX
CONFIG_KEYSTONE_ADMIN_TOKEN=92789c03f575453a91df6157246a865f
CONFIG_KEYSTONE_ADMIN_PW=XXXXXXXX
CONFIG_KEYSTONE_DEMO_PW=XXXXXXXX
CONFIG_KEYSTONE_TOKEN_FORMAT=PKI
CONFIG_GLANCE_DB_PW=XXXXXXXX
CONFIG_GLANCE_KS_PW=XXXXXXXX
CONFIG_CINDER_DB_PW=XXXXXXXX
CONFIG_CINDER_KS_PW=XXXXXXXX
CONFIG_CINDER_BACKEND=lvm
CONFIG_CINDER_VOLUMES_CREATE=y
CONFIG_CINDER_VOLUMES_SIZE=100G
CONFIG_CINDER_GLUSTER_MOUNTS=
CONFIG_CINDER_NFS_MOUNTS=
CONFIG_NOVA_DB_PW=XXXXXXXX
CONFIG_NOVA_KS_PW=XXXXXXXX
CONFIG_NOVA_SCHED_CPU_ALLOC_RATIO=16.0
CONFIG_NOVA_SCHED_RAM_ALLOC_RATIO=1.5
CONFIG_NOVA_COMPUTE_HOSTS=strežnik1,strežnik2
CONFIG_NOVA_COMPUTE_MIGRATE_PROTOCOL=tcp
CONFIG_NOVA_COMPUTE_PRIVIF=lo
CONFIG_NOVA_NETWORK_MANAGER=nova.network.manager.FlatDHCPManager
CONFIG_NOVA_NETWORK_PUBIF=bond0
CONFIG_NOVA_NETWORK_PRIVIF=lo
CONFIG_NOVA_NETWORK_FIXEDRANGE=<omrežje/maska> # Primer: 192.168.32.0/22
CONFIG_NOVA_NETWORK_FLOATRANGE=<omrežje/maska> # Primer: 10.3.4.0/22
CONFIG_NOVA_NETWORK_DEFAULTFLOATINGPOOL=nova
CONFIG_NOVA_NETWORK_AUTOASSIGNFLOATINGIP=n
CONFIG_NOVA_NETWORK_VLAN_START=100
CONFIG_NOVA_NETWORK_NUMBER=1
CONFIG_NOVA_NETWORK_SIZE=255
CONFIG_NEUTRON_KS_PW=XXXXXXXX
CONFIG_NEUTRON_DB_PW=XXXXXXXX
CONFIG_NEUTRON_L3_EXT_BRIDGE=br-ex
CONFIG_NEUTRON_L2_PLUGIN=ml2
CONFIG_NEUTRON_METADATA_PW=XXXXXXXX
CONFIG_LBAAS_INSTALL=y
CONFIG_NEUTRON_METERING_AGENT_INSTALL=n
CONFIG_NEUTRON_FWAAS=n
CONFIG_NEUTRON_ML2_TYPE_DRIVERS=vxlan
```

```
CONFIG_NEUTRON_ML2_TENANT_NETWORK_TYPES=vxlan
CONFIG_NEUTRON_ML2_MECHANISM_DRIVERS=openvswitch
CONFIG_NEUTRON_ML2_FLAT_NETWORKS=*
CONFIG_NEUTRON_ML2_VLAN_RANGES=
CONFIG_NEUTRON_ML2_TUNNEL_ID_RANGES=
CONFIG_NEUTRON_ML2_VXLAN_GROUP=
CONFIG_NEUTRON_ML2_VNI_RANGES=10:100
CONFIG_NEUTRON_L2_AGENT=openvswitch
CONFIG_NEUTRON_LB_TENANT_NETWORK_TYPE=local
CONFIG_NEUTRON_LB_VLAN_RANGES=
CONFIG_NEUTRON_LB_INTERFACE_MAPPINGS=
CONFIG_NEUTRON_OVS_TENANT_NETWORK_TYPE=vxlan
CONFIG_NEUTRON_OVS_VLAN_RANGES=
CONFIG_NEUTRON_OVS_BRIDGE_MAPPINGS=
CONFIG_NEUTRON_OVS_BRIDGE_IFACES=
CONFIG_NEUTRON_OVS_TUNNEL_RANGES=
CONFIG_NEUTRON_OVS_TUNNEL_IF=
CONFIG_NEUTRON_OVS_VXLAN_UDP_PORT=4789
CONFIG_HORIZON_SSL=n
CONFIG_SSL_CERT=
CONFIG_SSL_KEY=
CONFIG_SSL_CACHAIN=
CONFIG_SWIFT_KS_PW=XXXXXXXX
CONFIG_SWIFT_STORAGES=
CONFIG_SWIFT_STORAGE_ZONES=1
CONFIG_SWIFT_STORAGE_REPLICAS=1
CONFIG_SWIFT_STORAGE_FSTYPE=ext4
CONFIG_SWIFT_HASH=8f6fbc9fdeac4488
CONFIG_SWIFT_STORAGE_SIZE=2G
CONFIG_PROVISION_DEMO=y
CONFIG_PROVISION_TEMPEST=n
CONFIG_PROVISION_TEMPEST_USER=
CONFIG_PROVISION_TEMPEST_USER_PW=XXXXXXXX
CONFIG_PROVISION_DEMO_FLOATRANGE=
CONFIG_PROVISION_TEMPEST_REPO_URI=https://github.com/openstack/tempest.git
CONFIG_PROVISION_TEMPEST_REPO_REVISION=master
CONFIG_PROVISION_ALL_IN_ONE_OVS_BRIDGE=y
CONFIG_HEAT_DB_PW= XXXXXXXX
CONFIG_HEAT_AUTH_ENC_KEY= XXXXXXXX
CONFIG_HEAT_KS_PW= XXXXXXXX
CONFIG_HEAT_CLOUDWATCH_INSTALL=n
CONFIG_HEAT_USING_TRUSTS=y
CONFIG_HEAT_CFN_INSTALL=n
CONFIG_HEAT_DOMAIN=heat
CONFIG_HEAT_DOMAIN_ADMIN=heat_admin
CONFIG_HEAT_DOMAIN_PASSWORD=XXXXXXXX
CONFIG_CEILOMETER_SECRET=XXXXXXXX
CONFIG_CEILOMETER_KS_PW=XXXXXXXX
CONFIG_MONGODB_HOST= strežnik1
CONFIG_NAGIOS_PW=XXXXXXXX
##############################################################################
```

7.2 Skripta clean_ovs.sh

```
########################### Skripta clean_ovs.sh ############################
#!/bin/bash
#
# OPIS: Skripta počisti vse nastavitve, ki so nastale pri nameščanju
# oblačne platforme OpenStack.
#

for ns in `ip netns`
do
    `sudo ip netns del $ns`
done

for qvb in `ifconfig -a | grep qvb | cut -d' ' -f1`
do
    `sudo ip link set $qvb down`
    `sudo ip link delete $qvb`
done

for qbr in `ifconfig -a | grep qbr | cut -d' ' -f1`
do
    `sudo ip link set $qbr down`
    `sudo ip link delete $qbr`
done

for qvo in `ifconfig -a | grep qvo | cut -d' ' -f1`
do
    `sudo ovs-vsctl --if-exists del-port br-int $qvo`
done

for tap in `ifconfig -a | grep tap | cut -d' ' -f1`
do
    tap="${tap%?}"
    `sudo ip link set $tap down`
    `sudo ovs-vsctl --if-exists del-port br-int $tap`
done

for i in `sudo ovs-vsctl show | grep Bridge | awk '{print $2}'` ; do
    if [[ $i == *br-eth1* ]]; then
        sudo ovs-vsctl --if-exists del-br 'br-eth1'
    else
        sudo ovs-vsctl --if-exists del-br $i
    fi
done

for i in `ip addr | grep tap | awk '{print $2}'`; do
    tap="${i%?}"
    echo "tap=$tap"
    sudo ip link del dev $tap
done
```

```
for i in phy-br-eth1 int-br-eth1; do
    ip -o link show dev $i &> /dev/null
    if [ $? -eq 0 ]; then
        sudo ip link del dev $i
    fi
done

for iface in br-ex br-int br-tun; do
    sudo ovs-dpctl del-if ovs-system $iface
done

echo "Ce obstaja, izbriši vxlan"

for iface in `sudo ovs-dpctl show | awk 'match($0,
/[Pp]ort\s+[[:digit:]]+\s*\:\s*(.+).+\(vxlan/, m) { print m[1]; }'` ; do
   echo ${iface} ; sudo ovs-dpctl del-if ovs-system ${iface}
done
#############################################################################
```

7.3 Skripta conf_ovs_opendaylight.sh

```
#################### Skripta conf_ovs_opendaylight.sh ####################
#!/bin/bash
#
# OPIS: Skripta nastavi stikalo OVS tako, da komunicira s kontrolnikom ODL.
#

if [ `whoami` != "root" ]; then
       echo "Prosim zazenite sledeco skripto kot root ali z ukazom sudo z\
ustreznimi pravicami. "
exit 1
fi

if [ "$#" -ne 1 ]; then
        echo "Uporaba: conf_ovs_opendaylight.sh <tunel-ip_naslov>
<ODL_IP_naslov>"\ >&2
        echo " <tunel-ip_naslov> je kot local-ip nastavljen kot je  ovs-
neutron\
agent v datoteki ovs_neutron_plugin.ini"
exit 1
fi

ovs-vsctl set-manager tcp:$2:6640
read ovstbl <<< $(ovs-vsctl get Open_vSwitch . _uuid)
#ovs-vsctl set Open_vSwitch $ovstbl other_config={"local_ip"="$1"}
sudo ovs-vsctl set Open_vSwitch $ovstbl
other_config:bridge_mappings=physnet1:eth1,physnet3:eth3
sudo ovs-vsctl set Open_vSwitch $ovstbl other_config:local_ip=$1
ovs-vsctl list Open_vSwitch .
#############################################################################
```

7.4 Skripta set_of_rules.sh

```
######################### Skripta set_of_rules.sh ##########################
#!/bin/bash
#
# OPIS: Skripta kot vhodni parameter vzame datoteko, v kateri so nanizane
# zahteve za izvedbo akcij po protokolu HTTP. Parameter kontrolnika ODL
# priredite potrebam svojega kontrolnika.

if [[ $# -eq 0 ]] ; then
    echo 'Use script: ./set_of_rules service.txt'
    exit 0
fi

USER=admin
PASSWD=admin
CURL=$(which curl)
JSON_OPT="-H \"Accept: application/json\" -H \"Content-type:
application/json\""
ODL_IP=<IP_ODL_Kontrolnika>
ODL_PORT=8080
REQ_LIST=$1

IFS=$'\n'
for i in $(cat $REQ_LIST|grep -v \#|grep -v ^$)
do
   HTTP_REQ=$(echo $i | cut -d'!' -f1)
   VTN_ACTION=$(echo $i | cut -d'!' -f2)
   VTN_COMMAND=$(echo "$CURL --user \"$USER\":\"$PASSWD\" $JSON_OPT -X
$HTTP_REQ \
http://$ODL_IP:$ODL_PORT$VTN_ACTION" | sed 's/!/ /g')
$VTN_COMMAND
done
##############################################################################
```

7.5 Datoteka kreiranje_VTN_infra.txt

```
######################### kreiranje_VTN_infra.txt ##########################
#
# Datoteka je v formatu [POST|GET|PUT|DELETE]![VIR_REST]. Znak !
# (klicaj) pa se uporablja, kot ločilo med http zahtevami in
# indentifikatorij virov.
#

### Kreiranje VTN-okolja TenantTEST
POST!/controller/nb/v2/vtn/default/vtns/TenantTEST -d '{"description": "VN
za TenantTEST","idleTimeout": "0","hardTimeout": "0"}'

### Kreiranje navideznega stikala vBridge1 na TenantTEST
POST!/controller/nb/v2/vtn/default/vtns/TenantTEST/vbridges/Bridge1 -d
'{"description": "Virtaul Bridge1","ageInterval": "600"}'
```

```
### Kreiranje vmesnika IF1 na Bridge1 mostu
POST!/controller/nb/v2/vtn/default/vtns/TenantTEST/vbridges/Bridge1/interfa
ces/IF1 -d '{"description": "Vmesnik IF-1","enabled": true}'

### Povezava odjemalca 1 na vmesnik IF1
PUT!/controller/nb/v2/vtn/default/vtns/TenantTEST/vbridges/Bridge1/interfac
es/IF1/portmap -d '{"vlan": 0,"node": {"type": "OF","id":
"00:00:00:00:00:00:00:01"},"port": {"type": "OF","id": "2","name": "s1-
eth2"}}'

### Kreiranje vmesnika IF2 na Bridge1 mostu
POST!/controller/nb/v2/vtn/default/vtns/TenantTEST/vbridges/Bridge1/interfa
ces/IF2 -d '{"description": "Vmesnik IF-2","enabled": true}'

### Povezava Odjemalca 2 na most Bridge1
PUT!/controller/nb/v2/vtn/default/vtns/TenantTEST/vbridges/Bridge1/interfac
es/IF2/portmap -d '{"vlan": 0,"node": {"type": "OF","id":
"00:00:00:00:00:00:00:02"},"port": {"type": "OF","id": "2","name": "s2-
eth2"}}'

### Kreiranje vmesnika IF3 na mostu Bridge1
POST!/controller/nb/v2/vtn/default/vtns/TenantTEST/vbridges/Bridge1/interfa
ces/IF3 -d '{"description": "Opis IF-3","enabled": true}'

### Povezava Odjemalca 3 na most Bridge1
PUT!/controller/nb/v2/vtn/default/vtns/TenantTEST/vbridges/Bridge1/interfac
es/IF3/portmap -d '{"vlan": 0,"node": {"type": "OF","id":
"00:00:00:00:00:00:00:02"},"port": {"type": "OF","id": "3","name": "s2-
eth3"}}'
##############################################################################
```

7.6 Datoteka filter_odjemalec2.txt

```
########################## filter_odjemalec2.txt ##########################
#
# Datoteka je v formatu [POST|GET|PUT|DELETE]![VIR_REST]. Znak !
# (klicaj) pa se uporablja, kot ločilo med http zahtevami in
# indentifikatorji virov.
#
### Sledi filtriranje prometa
### Pogoj za tok do destinacije Odjemalec 2

PUT!/controller/nb/v2/vtn/default/flowconditions/pogoj_storitev -d
'{"name": "pogoj_odjemalec2","match": [{"index": 1,"inetMatch": {"inet4":
{"dst": "10.0.0.4"}}}]}'

### Tokovno filtreranje
PUT!/controller/nb/v2/vtn/default/vtns/TenantTEST/vbridges/Bridge1/interfac
es/IF1/flowfilters/in/10 -d '{"index": 10,"condition":
"pogoj_odjemalec2","filterType": {"drop": {}}}'
##############################################################################
```

7.7 Datoteka povezovanje_na_storitev1.txt

```
######################## povezovanje_na_storitev1.txt ######################
#
# Datoteka je v formatu [POST|GET|PUT|DELETE]![VIR_OBJEKTA]. Znak !(klicaj)
# pa se uporablja kot ločilo.
#

### Povezovanje s storivijo latence 200 ms
### Kreiranje vTerminala vt_streznik1_1
POST!/controller/nb/v2/vtn/default/vtns/TenantTEST/vterminals/vt_streznik1_
1 -d '{"description": "v/ za storitve1"}'

### Kreiranje vmesnika vTerminal IF
POST!/controller/nb/v2/vtn/default/vtns/TenantTEST/vterminals/vt_streznik1_
1/interfaces/IF -d '{"description": "Vmesnik IF-1","enabled": true}'

### Povezava
PUT!/controller/nb/v2/vtn/default/vtns/TenantTEST/vterminals/vt_streznik1_1
/interfaces/IF/portmap -d '{"vlan": 0,"node": {"type": "OF","id":
"00:00:00:00:00:00:00:03"},"port": {"type": "OF","id": "3","name": "s3-
eth3"}}'

### Kreiranje vTerminala vt_streznik1_2
POST!/controller/nb/v2/vtn/default/vtns/TenantTEST/vterminals/vt_streznik1_
2 -d '{"description": "vterminal za storitev1_2"}'

### Kreiranje vmesnika vTerminal IF
POST!/controller/nb/v2/vtn/default/vtns/TenantTEST/vterminals/vt_streznik1_
2/interfaces/IF -d '{"description": "Vmesnik IF-1_2","enabled": true}'

### Povezava
PUT!/controller/nb/v2/vtn/default/vtns/TenantTEST/vterminals/vt_streznik1_2
/interfaces/IF/portmap -d '{"vlan": 0,"node": {"type": "OF","id":
"00:00:00:00:00:00:00:04"},"port": {"type": "OF","id": "3","name": "s4-
eth3"}}'

### Postavitev pogoja za destinacijo odjemalca 2
PUT!/controller/nb/v2/vtn/default/flowconditions/pogoj_storitev -d
'{"name": "pogoj_storitev","match": [{"index": 1,"inetMatch": {"inet4":
{"dst": "10.0.0.4"}}}]}'

### Postavitev pogoja, ki velja za vse
PUT!/controller/nb/v2/vtn/default/flowconditions/pogoj_vsi -d '{"name":
"pogoj_vsi","match": [{"index": 1}]}'

### Flitriranje toka od vt_streznik1_2 do Bridge1-IF2
PUT!/controller/nb/v2/vtn/default/vtns/TenantTEST/vterminals/vt_streznik1_2
/interfaces/IF/flowfilters/in/10 -d '{"index": 10,"condition":
"pogoj_vse","filterType": {"redirect": {"destination": {"bridge":
"Bridge1","interface": "IF2"},"output": true}}}'

### Flitriranje toka od Bridge1-IF1 do vt_streznik1_1
PUT!/controller/nb/v2/vtn/default/vtns/TenantTEST/vbridges/Bridge1/interfac
es/IF1/flowfilters/in/10 -d '{"index": 10,"condition":
"pogoj_storitev","filterType": {"redirect": {"destination": {"terminal":
"vt_streznik1_1","interface": "IF"},"output": true}}}'
##############################################################################
```

7.8 Datoteka povezovanje_na_storitev2.txt

```
######################## povezovanje_na_storitev2.txt ######################
#
# Datoteka je v formatu [POST|GET|PUT|DELETE]![VIR_REST]. Znak !
# (klicaj) pa se uporablja kot ločilo med http zahtevami in
# indentifikatorji virov.
#

### Dodamo povezovanje s storivijo latence 200 ms
### Kreiranje vTerminala vt_streznik2_1
POST!/controller/nb/v2/vtn/default/vtns/TenantTEST/vterminals/vt_streznik2_
1 -d '{"description": "vterminal za storitev 2"}'

### Kreiranje vmesnika na vterminalu IF 2
POST!/controller/nb/v2/vtn/default/vtns/TenantTEST/vterminals/vt_streznik2_
1/interfaces/IF -d '{"description": "Vmesnik IF-2","enabled": true}'

### Povezovanje
PUT!/controller/nb/v2/vtn/default/vtns/TenantTEST/vterminals/vt_streznik2_1
/interfaces/IF/portmap -d '{"vlan": 0,"node": {"type": "OF","id":
"00:00:00:00:00:00:00:03"},"port": {"type": "OF","id": "4","name": "s3-
eth4"}}'

### Kreiranje vTerminala vt_streznik1_2
POST!/controller/nb/v2/vtn/default/vtns/TenantTEST/vterminals/vt_streznik2_
2 -d '{"description": "vterminal za service 2"}'

### Kreiranje vmesnika na vterminalu IF 2_2
POST!/controller/nb/v2/vtn/default/vtns/TenantTEST/vterminals/vt_streznik2_
2/interfaces/IF -d '{"description": "Vmesnik IF-2_2","enabled": true}'

### Povezava
PUT!/controller/nb/v2/vtn/default/vtns/TenantTEST/vterminals/vt_streznik2_2
/interfaces/IF/portmap -d '{"vlan": 0,"node": {"type": "OF","id":
"00:00:00:00:00:00:00:04"},"port": {"type": "OF","id": "4","name": "s4-
eth4"}}'

### Tokovno filtriranje od vt_streznik2_2 do vmesnika Bridge1-IF2
PUT!/controller/nb/v2/vtn/default/vtns/TenantTEST/vterminals/vt_streznik2_2
/interfaces/IF/flowfilters/in/10 -d '{"index": 10,"condition":
"pogoj_vsi","filterType": {"redirect": {"destination": {"bridge":
"Bridge1","interface": "IF2"},"output": true}}}'

### Tokovno filtriranje od vt_streznik1_2 do vterminala vt_streznik2_1
PUT!/controller/nb/v2/vtn/default/vtns/TenantTEST/vterminals/vt_srvc1_2/int
erfaces/IF/flowfilters/in/10 -d '{"index": 10,"condition":
"pogoj_vse","filterType": {"redirect": {"destination": {"terminal":
"vt_streznik2_1","interface": "IF"},"output": true}}}'
#############################################################################
```

7.9 Datoteka izbrisi_povezavo_do_odjemalca2.txt

```
##################### izklop_vmesnika_do_odjemalca2.txt ####################
#
# Datoteka je v formatu [POST|GET|PUT|DELETE]![VIR_REST]. Znak !
# (klicaj) pa se uporablja kot ločilo med http zahtevami in
# indentifikatorji virov.
#
### Izklop vmesnika do odjemalca 2

PUT!/controller/nb/v2/vtn/default/vtns/Tenant2/vbridges/Bridge1/interfaces/
IF2 -d '{"description": "Description about IF-2","enabled": false}'
##############################################################################
```

7.10 Datoteka kreiranje_povezavo_do_odjemalca2.txt

```
##################### vklop_vmesnika_do_odjemalca2.txt ######################
#
# Datoteka je v formatu [POST|GET|PUT|DELETE]![VIR_REST]. Znak !
# (klicaj) pa se uporablja kot ločilo med http zahtevami in
# indentifikatorji virov.
#
### Vklop vmesnika do odjemalca 2

PUT!/controller/nb/v2/vtn/default/vtns/Tenant2/vbridges/Bridge1/interfaces/
IF2 -d '{"description": "Description about IF-2","enabled": true}'
##############################################################################
```

Literatura

[1]. Daniel Nurmi, Rich Wolski, Chris Grzegorczyk Graziano Obertelli, Sunil Soman, Lamia Youseff, Dmitrii Zagorodnov, »The Eucalyptus Open-source Cloud-computing System«, *CCGRID '09 Proceedings of the 9th IEEE/ACM International Symposium on Cluster Computing and the Grid (IEEE Computer Society Washington)*, str. 124-131, 2009.

[2]. Peter Sempolinski, Douglas Thain, »A Comparison and Critique of Eucalyptus, OpenNebula and Nimbus«, *CLOUDCOM '10 Proceedings of the IEEE Second International Conference on Cloud Computing Technology and Science*, str. 417-426, 2010.

[3]. Official Documentation for Eucalyptus Cloud. Dostopno na: https://www.eucalyptus.com/docs/eucalyptus/4.1.1/index.html (pridobljeno 10. 6. 2015)

[4]. Official OpenNebula documentation. Dostopno na: http://opennebula.org/documentation/ (pridobljeno 10. 6. 2015)

[5]. Official CloudStack documentation. Dostopno na: http://docs.cloudstack.apache.org/en/master/ (pridobljeno 10. 6. 2015)

[6]. Official OpenStack documentation. Dostopno na: http://docs.openstack.org/ (pridobljeno 10. 6. 2015)

[7]. Sonali Yadav, »Comparative Study on Open Source Software for Cloud Computing Platform: Eucalyptus, Openstack and Opennebula«, *International Journal Of Engineering And Science,* zv. 3, št. 10*,* 2013.

[8]. Rakesh Kumar, Kanishk Jain , Hitesh Maharwal, Neha Jain , Anjali Dadhich, »Apache CloudStack: Open Source Infrastructure as a Service Cloud Computing Platform«, *International Journal Of Engineering And Science*, zv.1 št. 2, 2014.

[9]. Thomas. D. Nadeau, Ken Gray, SDN: Software Defined Networks, *O'Reilly Media, Inc*, ISBN:978-1-4493-4230-2, 2013.

[10]. Susan Hares, Russ White, »Software-Defined Networks and the Interface to the Routing System (I2RS)«, *IEEE Educational Activities Department Piscataway,* št. 17, zv. 4, str. 84-88, 2013.

[11]. Nick Feamster, Jennifer Rexford, Ellen Zegura, »The Road to SDN: An Intellectual History of Programmable Networks«, *ACM SIGCOMM Computer Communication Review table of contents archive*, zv. 44 št. 2, str. 87-98, 2014.

[12]. Stefan Schmid, Jukka Suomela, »Exploiting Locality in Distributed SDN Control«, *ACM SIGCOMM workshop on Hot topics in software defined networking*, str. 121-126, August 2013.

[13]. Amin Tootoonchian, Sergey Gorbunov, Yashar Ganjali, Martin Casado, Rob Sherwood, »On Controller Performance in Software-Defined Networks«, *Proceedings of the 2nd USENIX Association Berkeley conference on Hot Topics in Management of Internet, Cloud, and Enterprise Networks and Service,* str. 10-10, 2012.

[14]. Theophilus Benson, Aditya Akella, Anees Shaikh, Sambit Sahu, »CloudNaaS: A Cloud Networking Platform for Enterprise Applications«, *SOCC '11 2nd ACM Symposium on Cloud Computing Article* št. 8, ACM 2011.

[15]. Michael Jarschel, Simon Oechsner, Daniel Schlosser, Rastin Pries, Sebastian Goll, Phuoc Tran-Gia, »Modeling and Performance Evaluation of an OpenFlow Architecture«, *ITC '11 23rd International Teletraffic Congress,* str. 1-7, ITCP 2011.

[16]. OpenFlow Switch Specification, Open Network Faundation, 2014. Dostopno na: https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-switch-v1.5.0.noipr.pdf

[17]. Sebastian Dabkiewicz, Ronald van der Pol, Gerben van Malenstein, »OpenFlow network virtualization with FlowVisor«, *Research project 2, University of Amsterdam*, November 2012.

[18]. R. Enns, M. Bjorklund, J. Schoenwaelder, A. Bierman, »RFC 6241: Network Conguration Protocol (NETCONF)«, *IETF*, 2011.

[19]. T. Koponen, K. Amidon, P. Balland, M. Casado, A. Chanda, B. Fulton, I. Ganichev, J. Gross, N. Gude, P. Ingram, E. Jackson, A. Lambeth, R. Lenglet, S.-H. Li, A. Padmanabhan, J. Pettit, B. Pfaff, R. Ramanathan, S. Shenker, A. Shieh, J. Stribling, P. Thakkar, D. Wendlandt, A. Yip, R. Zhang, »Network virtualization in multi-tenant datacenters«, *USENIX Association Berkeley*, str. 203-216, 2014.

[20]. B. Pfaff, J. Pettit, T. Koponen, E. J. Jackson, A. Zhou, J. Rajahalme, J. Gross, A. Wang, J. Stringer, P. Shelar, K. Amidon, M. Casado, »The Design and Implementation of Open vSwitch«, *USENIX Association Berkeley,* 2015.

[21]. Rodrigues Prete, L. Fac. de Tecnol. de Jales, Centro Estadual de Educ. Tecnol. Paula Souza, Jales, Brazil Schweitzer, C.M. ; Shinoda, A.A. ; Santos de Oliveira, R.L., »Simulation in an SDN network scenario using the POX Controller«, *IEEE Colombian Conference Communications and Computing (COLCOM)*, str. 1-6, 2014.

[22]. R. Khondoker, A. Zaalouk, R. Marx, K. Bayarou, »Feature-based comparison and selection of Software Defined Networking (SDN) controllers*«, International Conference on Computer Software and Applications,* 2014.

[23]. Ryan Wallner, Robert Cannistra, Marist College, »An SDN Approach: Quality of Service using Big Switch's Floodlight Open-source Controller*«, Proceedings of the Asia-Pacific Advanced Network*, zv. 35, str. 14-19, 2013.

[24]. OpenDaylight Wiki portal. Dostopno na: https://wiki.opendaylight.org/view/Main_Page (pridobljeno 11. 6. 2015)

[25]. Kreutz, D., Ramos, F.M.V., Esteves Verissimo, P., Esteve Rothenberg, C., Azodolmolky, S., Uhlig, S, »Software-Defined Networking: A Comprehensive Survey«, *IEEE*, zv. 103, ver. 1, str. 14-76, 2015.

[26]. OSGi Alliance. pridobljeno na: http://www.osgi.org/Main/HomePage (pridobljeno 11. 6. 2015)

[27]. OpenDaylight Foundation, »Technical Overview«. Dostopno na: http://www.opendaylight.org/project/ (pridobljeno 11. 6. 2015)

[28]. Linux Foundation, https://wiki.opendaylight.org/view/OpenDaylight_Controller:MD-SAL:MD-SAL_App_Tutorial,
(pridobljeno 11. 6. 2015)

[29]. Linux Foundation, Dostopno na:
https://wiki.opendaylight.org/view/OpenDaylight_Controller:MD-SAL:Model_Reference,
(pridobljeno 11. 6. 2015)

[30]. OpenDaylight ProjectProposals:Defense4All. Dostopno na:
https://wiki.opendaylight.org/view/Project_Proposals:Defense4All
(pridobljeno 11. 6. 2015)

[31]. RDO project. Dostopno na: https://www.rdoproject.org/Docs
(pridobljeno 11. 6. 2015)

[32]. Tom Fifield, Diane Fleming, Anne Gentle, Lorin Hochstein, Jonathan Proulx, Everett Toews, Joe Topjian »OpenStack Operations Guide«, *O'Reilly Media, Inc.,* ISBN 1491946954 9781491946954, 2014.

[33]. OpenStack Operations Guide« https://www.rdoproject.org/Docs
(pridobljeno 11. 6. 2015)

[34]. Mian, A.N., Mamoon, A. ; Khan, R., Anjum, A, »Effects of Virtualization on Network and Processor Performance Using Open vSwitch and Xen Server« *Utility and Cloud Computing (UCC), 2014 IEEE/ACM 7th International Conference*, str. 762-767, 2014.

[35]. Open vSwitch Advanced Features Tutorial. Dostopno na:
https://github.com/openvswitch/ovs/blob/master/tutorial/Tutorial.md
(pridobljeno 11. 6. 2015)

[36]. OpenDaylight Virtual Tenant Network. Dostopno na:
https://wiki.opendaylight.org/view/OpenDaylight_Virtual_Tenant_Network_(VTN):Main
(pridobljeno 11. 6. 2015)

[37]. OpenDaylight User guide Dostopno na:
https://wiki.opendaylight.org/view/Release/Helium/VTN/User_Guide
(pridobljeno 11. 6. 2015)

[38]. Doug Marschke, Jeff Doyle, Pete Moyer, »Software Defined Networking (SDN): Anatomy of OpenFlow«, *Lulu publishing services*, zv. 1, 2015.

[39]. David Erickson, »The Beacon OpenFlow Controller«, *Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking*, str. 13 – 18, 2013.

[40]. Peter Mell, Timothy Grance »The NIST Definition of Cloud Computing«, *NIST Special Publication 800-145*, 2011.

[41]. Big Network Controller. Dostopno na:
http://www.bigswitch.com/products/SDN-Controller
(pridobljeno 20. 6. 2015)